Northumbria Research Link

Citation: Eliot, Neil (2017) Methods for the Efficient Deployment and Coordination of Swarm Robotic Systems. Doctoral thesis, Northumbria University.

This version was downloaded from Northumbria Research Link: http://nrl.northumbria.ac.uk/id/eprint/32575/

Northumbria University has developed Northumbria Research Link (NRL) to enable users to access the University's research output. Copyright © and moral rights for items on NRL are retained by the individual author(s) and/or other copyright owners. Single copies of full items can be reproduced, displayed or performed, and given to third parties in any format or medium for personal research or study, educational, or not-for-profit purposes without prior permission or charge, provided the authors, title and full bibliographic details are given, as well as a hyperlink and/or URL to the original metadata page. The content must not be changed in any way. Full items must not be sold commercially in any format or medium without formal permission of the copyright holder. The full policy is available online: http://nrl.northumbria.ac.uk/policies.html

Northumbria University NEWCASTLE

CUSTOMER SERVICE EXCELLENCE CSE

UniversityLibrary

METHODS FOR THE EFFICIENT DEPLOYMENT AND COORDINATION OF SWARM ROBOTIC SYSTEMS

NEIL ELIOT

PhD

2017

METHODS FOR THE EFFICIENT DEPLOYMENT AND COORDINATION OF SWARM ROBOTIC SYSTEMS

NEIL ELIOT

A thesis submitted in partial fulfilment
of the requirements of the
University of Northumbria at Newcastle
for the degree of
Doctor of Philosophy

Research undertaken in the Faculty of
Engineering and Environment

January 2017

ABSTRACT

Swarming has been observed in many animal species, including fish, birds, insects and mammals. These biological observations have inspired mathematical models of distributed coordination that have been applied to the development of multi-agent robotic systems, such as collections of unmanned autonomous vehicles (UAVs). The advantages of a swarming approach to distributed coordination are clear: each agent acts according to a simple set of rules that can be implemented on resource-constrained devices, and so it becomes feasible to replicate agents in order to build more resilient systems. However, there remain significant challenges in making the approach practicable. This thesis addresses two of the most significant: coordination and scalability. New coordination algorithms are proposed here, all of which manage the problem of scalability by requiring only local proximity sensing between agents, without the need for any other communications infrastructure.

A major source of inefficiency in the deployment of a swarm is 'oscillation': small movements of agents that arise as a side effect of the application of their rules but which are not strictly necessary in order to satisfy the overall system function. The thesis introduces a new metric for 'oscillation' that allows it to be identified and measured in swarm control algorithms.

A new perimeter detection mechanism is introduced and applied to the coordination of goal-based swarms. The mechanism is used to improve the internal coordination of agents whilst maintaining a directional focus to the swarm; this is then analysed using the new metric.

A mechanism is proposed to allow a swarm to exhibit a 'healing' behaviour by identifying internal perimeter edges (doughnuts) and then altering the movement of agents, based upon a simple criterion, to remove the holes; this also has the emergent effect of smoothing the outer edges of a swarm and creating a more uniform swarm structure.

Area coverage is an important requirement in many swarm applications. Two new, efficient area-filling techniques are introduced here and exit conditions are identified to determine when a swarm has filled an area.

In summary, the thesis makes significant contributions to the analysis and design of efficient control algorithms for the coordination of large-scale swarms.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my Mother for the mantra of 'Keep on learning!' I will be forever grateful.

I would like to thank my Wife who is both an inspiration and a pillar to my existence, without her there would be no me!

I would also like to thank my Supervisory team (Dr David Kendall, Dr Michael Brockway and Prof. Ahmed Bouridane) without whom none of this would have been possible. I hope that the University appreciate their dedication to teaching and research and bestow upon them sabbaticals and research time so they may influence others as they have influenced me.

I would also like to thank others who have influenced me at different stages in my life, to mention but a few, Bill (William) Henderson, John Eakins and Adrian Robson and from secondary school Mr (Dominic) James an inspirational teacher!

DECLARATION

I declare that the work contained in this thesis has not been submitted for any other award and that it is all my own work. I also confirm that this work fully acknowledges opinions, ideas and contributions from the work of others. Any ethical clearance for the research presented in this thesis has been approved. Approval has been sought and granted by the Faculty Ethics Committee on 3rd March 2016.

Word count: 44,876

Name: Neil Eliot

Signature:

Date:

1. INTRODUCTION AND OVERVIEW

Swarming in the animal kingdom of ants, bees, fish and birds for instance has long been studied by scientists. From these studies mathematical models and algorithms have evolved. The models and algorithms have in turn captured the interest of computing scientists who are interested in applying them to large groups of autonomous mobile *agents* ('robots'). The cooperative coordination of these agents can take many forms such as following a set path [62], existing in a static space [32, 39, 40] or foraging as a colony [55, 47]. One of the attributes of swarms that has captured the interest of scientists is that the models and algorithms used to coordinate them are generally sets of simple rules. These simple rules cause the agents to appear to work cooperatively. Swarms can also exhibit features or behaviours which were not expected. This is due to the global effects of the simple algorithms being executed in a distributed manner. These unexpected behaviours are known as *'emergent behaviours'* [125, 126].

The ability to have autonomous agents working collaboratively has led to the development of systems that use this phenomenon to solve problems in different ways. In 1986, before swarming was being widely used as a technology, there was an explosion at the Chernobyl nuclear power station. To determine the extent of the destruction, a robot was deployed to inspect the reactor base and carry out surveillance of the damage to the building [1]. The robot was manually operated and had no autonomous capabilities. More recently, in 2015, a project was undertaken to carry out surveillance of forest fire perimeters [17]. The difference between these two surveillance projects was that rather than employing a single robot to carry out the surveillance of the forests, a swarm of *decentralised* autonomous agents was deployed. This illustrates a developing trend of applying swarms to the problem of environmental surveillance.

1.1 Biological swarms

Swarming has been identified in many species including fish, birds, insects, and more recently, mammals [146]. It is believed that this behaviour has evolved over thousands of

years, through natural selection, as a mechanism to improve the survival of species [166].

Fish swarm in the form of shoals [112] in an attempt to make it more difficult for predators to catch them. It is thought that grouping together makes it difficult for a predator to isolate an individual [88].

Birds flock together for the same reason as fish, to increase their survival prospects, but also to improve the efficiency of area coverage when feeding [112]. In the case of starlings and their evening murmurings [162], it is believed that the flock is identifying an optimal roost for itself, while ensuring its survival by disorientating predators. The disorientation for the predator stems from the distribution of the individuals in the flock as it moves [20].

Locusts swarm when feeding to make best use of the food resource by increasing the coverage of an area to ensure the resources are exhausted [48, 148].

Ants and bees live in colonies [73, 135] and it is believed they swarm to make best use of their resources and also to allow specialisations within their communities. The specialised individuals would not be able to survive alone but, as part of a colony, they add value to the group. Ants for instance have specialisations such as soldier ants for defence and nursing ants to look after pupae. In bee swarms there are workers that forage and queens and drones that remain in the hive [16].

More recently there has been research to show that swarming-based behaviours exist in higher order animals such as baboons [146], where they use a consensus-based decision process to determine a troupe's movement. Yao and Hwang have analysed human behaviour and found humans exhibit *boid-based* behaviours when in groups through *cohesion* (§ 2.4) and *repulsion* (§ 2.5) which they refer to as *separation*, the third component they discuss is *alignment* (§ 2.7), which is a consensus-based directional movement [161]. Reynolds [124] describes this same structure when describing *boid-based* movement.

All these adaptations and behaviours have led to the ecology community focusing on how these behaviours emerge and to use computer simulation to emulate the behaviour, and therefore understand the mechanisms the swarms use [34]. In the case of analysing baboons, they used high-accuracy GPS trackers and with humans they used phone based GPS data [146].

The general consensus is that nature, through natural selection, refines behaviours to sustain a population or to help it adapt or expand. This has led to research into the mechanics of how animals interact to achieve these swarming effects [34, 108, 148]. Passino

has analysed bee populations in a hive [135] and authored books on bio-mimicry. He has also authored several papers on the computational theory of swarm stability with Gazi [40, 39, 38]. This shows a link between the natural world and computer science.

All these naturally occurring swarms provide paradigms that allow the categorisation of swarms. Naturally occurring swarms include foraging-based (bees), colony-based (ants) and flocking-based (fish/birds). These basic swarming models have been used to influence how computational models are designed to mimic the behaviours found in nature. These models can be applied to robotic swarms which are used for specific tasks based upon behavioural requirements [82].

1.2 Computational swarms

Computational swarms are inspired by aspects of biological swarms. The degree to which the biological swarm is emulated within the computational environment varies. A prominent feature that is frequently emulated is the cooperative behaviour of the swarm agents by simulating agents movements using repulsion and cohesion between the agents. The emergent behaviours that simple algorithms create through these agent interactions is the focus of this thesis.

1.2.1 Foraging swarms

Foraging swarms are composed of agents that emulate the natural world by carrying out tasks that involve a permanent base. The tasks are carried out by agents to ensure the colony survives or expands. The coordination in these types of swarm is for the colony to maintain itself by using scout agents to locate resources that are required and then to return those resources to the colony [38, 58, 85, 84]. The foraging component of this process is the locating of resources. There is also a communications component to foraging swarms. Foraging agents inform the rest of the colony of the resource locations to optimise the foraging tasks. Beeclust [115, 55], Swarm-agent [97] and other bee-inspired algorithms [78] are all implementations of this type of swarming behaviour.

1.2.2 Ant-colony swarms

Ant-colony swarms [138, 96, 51, 89] are similar to foraging swarms in terms of their interactions. The difference is in the way the agents communicate with each other. The

agents move independently and there is no need for them to ensure proximity as the agents follow predefined routes. The agents are therefore independent in that there is no centralised coordinator and they act autonomously. The cooperation component of the swarm is realised by agents highlighting desirable or undesirable routes. The agents then follow the same trails back to a base and either reinforce or reduce the importance of the routes by adding or removing a pheromone [154]. The purpose of the trails are determined by identifying the needs of the colony centrally (at the base). In nature this is exactly how ant colonies function [69]. Some robotic ant-colony simulations include the concept of the 'pheromone decay' process as found in biological ant colonies [127]. This process allows for changes in the priorities of the agents to be based on time as well as reinforcement as a colony propagates through, or exists within, an environment.

1.2.3 Boid-based swarms

Boid-based swarms, as originally defined by Reynolds [124], are modelled on the behaviour of fish and birds. They are composed of autonomous agents that are *decentralised* and formulate their positions based upon an awareness of the location of their neighbours [68, 25, 57]. The agents in a boid-based swarm are independent and each control their own position. The two major factors that create the swarming effect are *cohesion* and *repulsion*. Cohesion ensures the swarm has a tendency to stay together as a single entity. This has been used in the SmartBot project, where it has been found that cohesion promotes the collaboration of autonomous agents [29, 30, 97]. Repulsion ensures that the agents do not collide, and when balanced with cohesion create a well-structured swarm. The balancing of these two factors is identified by Gazi and Passino in their swarm stability papers [40, 39, 38] and as part of the GUARDIAN project [130]. They also discuss cohesion and repulsion in their book on swarm optimisation [42].

A directional component can also be incorporated into the movement of agents. The movement can be based on the direction of an agent's neighbours as in [124, 68], where direction is referred to as alignment. Alignment is a consensus-based direction that the agents negotiate by communicating with each other. The negotiated direction is not based on a set goal that the swarm must move towards.

Direction can also be applied as a goal as discussed by Hiroshi et al. and Navarro et al. in [53, 103] where the direction is based on a position that the swarm must move towards and each agent is able to identify the direction locally. The goal can be determined based upon local environmental stimuli, such as temperature [113]. If a swarm is to be used in

an open air environment covering a large area, a GPS sensor can be used to determine its goal [131].

1.2.4 Centralised swarms

The concept of centralised coordination is not seen in biological swarms. Centralised swarms are inspired by the benefits of cooperative agents being used to solve a problem. The agents themselves are autonomous in terms of their function but their positional autonomy is removed and they are centrally managed. This centralised paradigm determines the type of tasks the swarm can be applied to [7, 96, 81]. Centralised swarms are deployed into a known environment and a central controller coordinates the positional information [65, 105, 144, 95]. The model calculates positional requirements for all the agents and transfers that information to the agents through a communications infrastructure. This is different to decentralised swarms, such as boid-based swarms, that are predominantly based upon localised proximity field effects [11, 10, 12, 14]. Field effects are the omni-directional ranges used by an autonomous agent to determine the proximity of nearby agents to determine their relationship [11]. Centralised swarms are different from swarms that use an internal communications infrastructure to negotiate roles and exchange information [107] such as the BEECLUST swarm [55].

In a centralised swarm the positioning of the agents is entirely determined by a central controller, and communicated to them by it. The controller is a single point of failure and the communications overheads can be significant [90]. This adversely affects the reliability and scalability of the swarm.

The central processing of the algorithm is complex due to calculating multiple agent locations rather than a single location. The processing complexity can be overcome by increasing the performance of the central controller, but this will not overcome the communications problem. This type of swarm works well when creating predefined structures such as the tower building swarm [45] or the knot tying quad-copters [7] that have been developed as part of the research projects of D'Andrea in the department of Dynamic Systems and Control at ETH Zurich. The focus of this thesis is on swarms that do not require a central controller. Control is distributed and each agent acts independently.

1.3 Swarming applications

Many industries require the exploration or reconnaissance of environments that are not easily accessible by humans. Consider for instance disaster areas following earthquakes, or environments that are simply hard to survey due to their size, such as large scale commercial farms [17, 21].

There are occasions on which it is necessary to explore underground or enclosed spaces [64]. In mining, for example, the environment may be a labyrinth of tunnels that may be dangerous due to rock falls or toxic gas etc. Such environments may consist of many rejoining routes and dead ends. This type of work is best performed by swarms of autonomous robots [82].

An example of a large implementation of a swarming platform is Project Loon [71, 44, 43]. Google have completed trials and are now creating aerial platforms with high altitude balloons to provide communication infrastructures in remote areas of New Zealand [106, 54]. There are also smaller scale projects investigating the use of swarms in surveying crops to check the health of plants [17]. This is to identify remedial actions that can improve crop yields. The forestry commission have carried out surveys of forest environments using swarms of UAVs (Unmanned Aerial Vehicles). All these applications require the agents not only to coordinate themselves within the swarm environment, but also to carry sensor arrays to detect environmental conditions.

There is a view that swarms can be made to interact with humans. In 2011, a trial took place that used swarms to assist individuals in the fire service [114] as part of the GUARDIAN project [130]. In 2005 there was research by Stormont into the use of swarms to assist homeland first responders [145]. The paper concluded that *"the RoboCup goal of fully autonomous collaborative rescue robots by 2050 is a pretty good estimate"*.

It is clear that the application of swarms has increased and diversified into many industries. This has been made possible by the increased understanding of their capabilities. The work in this thesis further increases that understanding.

1.4 Focus of the thesis

This thesis takes its lead from swarming in the natural world and focuses on boid-based swarms with the addition of a directional component where necessary. The directional

component will be applied as a global positional requirement of the swarm as used in large scale reconnaissance projects.

Although research into swarming algorithms can be carried out using both physical implementations [29, 30] and software simulations [11, 39], the work covered in this thesis uses only software simulations. This makes possible the study of very large swarms over flexible time scales.

The application of swarms to solve large scale problems has increased as greater understanding of how swarms can be coordinated and monitored has improved. This thesis describes the development of a new metric for evaluating configurable coordination algorithms. This increases the understanding of how the dynamics of a swarm can be tailored to specific application areas. The algorithms, metric, and simulator have been developed as part of this thesis.

The thesis argues that the utility of a swarm in reconnaissance can be improved by exploiting emergent behaviours to improve the area coverage of goal based swarms when encountering obstacles. This could improve detection rates when swarms are used for searching activities such as locating targets within a large area. These targets could be mountaineers in remote areas or livestock on commercial farms.

The thesis also identifies behaviours that can be used to promote a *self-healing* effect to improve the structure of a swarm [129]. Self-healing is the ability of a swarm to remove 'holes' from it's structure. This behaviour can also be applied to surrounding objects. The oil industry has been involved in several man-made disasters involving large scale oil spillages. Research into possible containment of these spillages has shown it is possible to use warms to identify an oil slick's perimeter [165]. This thesis shows that the self-healing effect can be applied to the task of containing an oil spillage.

1.5 Contributions

Navarro defined a set of metrics for the analysis of swarms [104]. These metrics were based on the positions of agents in a swarm and looked at average speed, density of the population, and variations in distances. This thesis proposes a new metric for swarm analysis.

The new metric is based upon the inter-agent interactions and is independent of the distribution of the agents. The interactions are the magnitudes of the cohesion and

repulsion vectors that the field effects and algorithms produce. These same vectors when summed and normalised produce the directional vector of each agent. By focusing on the interaction of the agents at the mathematical layer rather than just the spatial distribution the metric identifies the degree of influence each agent has upon its neighbours. The inter-agent interactions can be used as a comparative metric for different swarming algorithms. The new metric identifies the effects of different algorithms when they produce both regular and irregular spatial distributions. The metric can also be used to highlight specific states in a swarm such as when flood filling an area. i.e. the inter-agent magnitude increases without causing a spatial distance increase. This state identification can be used as an exit condition for an area filling task.

This thesis introduces three directional algorithms that allow swarms to be applied to tasks such as search and rescue or reconnaissance. Most directional swarms use some kind of positioning system which all agents employ. This thesis demonstrates that it is possible to reduce the number of agents in a swarm employing a positioning system in a consistent manner such that the swarm still exhibits a directional bias. These new algorithms also reduce the gross energy consumption of the swarm making the swarm more energy efficient. The thesis also demonstrates that a reduction in the position system utilisation reduces the inter-agent disturbances.

This thesis demonstrates that emergent behaviours can be exploited to improve the structure of a swarm. Swarms, by consisting of many agents, are resistant to agent failures. However failures can occur and when they do they create gaps in the swarm's structure where the failed agents were located. Swarms can also develop irregularly shaped perimeters with dents. Dents are concave deformations caused by deployment irregularities, external effects such as obstacles, or perimeter agents coming into contact with additional agents. These characteristics (anomalies) reduce the effectiveness of a swarm in some tasks due to the overall structure being non-uniform. This thesis addresses these specific issues by extending the basic swarming algorithm to produce a localised agent effect that has a global impact on the swarm's structure by reducing and removing these anomalies thus 'healing' the swarm.

Riano [125] has shown that there are hidden benefits in using swarms due to the emergent behaviours of group dynamics which can assist reconnaissance. Swarms are often modelled in environments that include obstacles that must be avoided [37, 8, 149]. These obstacles can cause voids in a swarm. A void is an area within the body of a swarm where there are no agents. The void reduction technique developed in this thesis increase the ability of a swarm to reduce voids that are created by an obstacle when they are in

the path of a goal based swarm. By using this 'healing' effect to remove the voids it is possible to increasing the 'coverage' around an obstacle. This builds upon the work by Geunho Lee and Nak Young Chong [77].

Arkin et al., Fang et al., Krothapalli et al. and Luc Moreau state that agents in a swarm need to communicate with one another locally in order to maintain a swarm's structure [5, 35, 74, 98, 117]. Hoff et al. have shown that localising communications to just neighbours is advantageous [58] as it reduce the message propagation pathways within a swarm. Nithin et al. state that agents could have a central communications infrastructure that is independent of the agents [90], as used in centralised swarms.

Alternatively Higgins et al. and Navarro et al. state that inter-agent communications limit or impair a swarm's functionality [56, 103]. This thesis proposes that a communications infrastructure is not required for the identification of features such as perimeters as local positional information is all that is required. Local positioning can be obtained without inter-agent communications by using sensors such as an omni-directional camera.

This thesis demonstrates algorithms that are able to detect perimeters, which are the edges of a swarm, and perimeter anomalies (deformations) without the need for a global swarm based communications infrastructure. This removal of the need for message propagation allows the algorithms to be applied to arbitrary sized swarms.

This thesis focuses on arbitrary sized swarms. Modelling large numbers of agents in a swarm is most practicably carried out using a simulator. The requirements of the swarm analysis using inter-agent interactions is a very specific requirement. Combining these two requirements a bespoke simulator is presented as part of this thesis. The simulator is designed using an object model approach with data capture and accurate modelling as the primary goals. The object model used in the simulator is similar to that described by Vankerkom and Yu [152] and provides an extensible framework for the development of swarming applications. This thesis uses the framework to create two applications. A graphical scenario creation tool and a command line simulation tool.

1.6 Structure of the thesis

The rest of the thesis is structured as follows: Chapter 2 covers methods tools and techniques used to implement the coordination of agents in a swarming structure. Chapter 3 covers the simulator that has been developed in order to investigate the algorithms

proposed as part of this thesis. Chapter 4 discusses the development and application of the metric that allows the analysis of the effect a particular swarming algorithm has on a swarm's internal movement. Chapter 5 presents the metric and shows how the metric can be used to identify the effects of algorithms and field effects on the structure of a swarm and how different inter-agent relationships can be identified. Chapter 6 discusses two methods of coordinating a goal-based swarm and a baseline for comparison. This chapter also identifies the changes these algorithms generate on the movements of agents within a swarm. Chapter 7 examines the emergent behaviours of void reduction on goal-based and stationary swarms. Chapter 8 discusses the use of field effects to create an area filling behaviour and demonstrates how the new metric can be used to identify an exit condition when the area filling is completed. Chapter 9 sums up all the findings of the thesis and identifies additional work that has been identified through the research carried out as part of this thesis.

2. METHODS, TECHNIQUES, TOOLS

This chapter introduces the representation of agents, swarms, obstacles, the environment, and the algorithms applied to inter-agent and obstacle interactions to produce a swarming effect. Movement of agents and the application of a *destination vector* for goal based swarms are presented [100].

2.1 Modelling agents and swarms

Currently, much swarm research uses field effects as the method of modelling inter-agent interactions [11, 10, 14, 3, 39, 40, 38, 41, 42, 95]. The models usually use two field effects to implement the swarming characteristic. These effects are *cohesion*, to draw agents closer, and *repulsion* to prevent agents colliding. Field effects are the ranges around an agent that determine the effect other agents have upon its movement (Figure 2.1). It is usual for the cohesion field to have a radius C_b which is larger than the repulsion radius R_b. When an agent (b') moves into the *neighbour field* of an agent (b) then b' is said to be a neighbour of b and is subject to cohesion. When an agent b' moves into the repulsion field of b then b has a tendency to move away from b', i.e. to be repulsed. When an agent b moves too close to an obstacle, i.e. within the obstacle repulsion range O_b, it has a tendency to move away from the obstacle.

A common approach to the application of field effects is to use fixed ranges common to all agents. Cohesion is applied graduated by neighbour proximity and repulsion is applied as a fixed magnitude when an agent is within the *repulsion field*.

Sensing devices have a limited range within which they detect agents, this determines a *sensing field* shown is black (Figure 2.1). In a physical implementation of a swarm, distance may be determined by some form of sensing device such as an omni-directional camera, as used in the s-bot project [60, 97, 93], or lidar [79] or ultrasonic sensors [22] or by an array of simple proximity detectors [59]

This thesis uses a similar approach to applying the cohesion effect but for repulsion a

graduated field effect based on neighbour agent proximity is used.

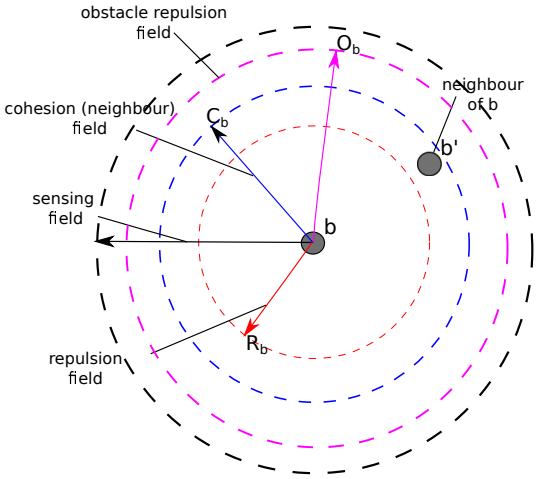


Fig. 2.1: Agent field effects

A swarm is modelled as a set of agents [93, 152]. An agent is modelled as a point in 2 dimensional space with no mass or size. This is similar to the representation used by Vankerkom and Yu to visualise swarms [152]. Mohan and Ponnambalam, and Gazi and Passino [152, 40], Barnes et al. [11, 10, 12] and Bennet and McInnes [14] and Andreou et al. [3] also use a similar model which includes agents moving at a constant speed.

The interaction of agents within a swarm is modelled using vectorial and geometric techniques [55, 11]. The position of an agent is modelled using cartesian coordinates and the movements are modelled using vectors. The position vector is given by the coordinates of an agent.

The 'world' that the swarm is modelled within is an unbounded 2 dimensional Euclidean plain.

The use of vectors to model inter-agent interactions is also referred to a artificial potential fields [37, 157, 133, 11, 10, 12, 14, 61] or vector fields [161, 49, 111].

2.2 Modelling agents and environment interactions

The environment contains agents with a position in the coordinate system. It may also contain *obstacles* (Figure 2.6) and *destinations* (Figure 2.7). An obstacle can be

considered as a point with an associated *repulsion field* effect and a destination as a point towards which agents move. This modelling technique is similar to that used by Barnes et al. [11, 10].

The distribution of each of these objects, along with the field effects, produce sets of vectors that represent the inter-object interactions in the system. The vector sets for each agent are used to calculate a vector for each interaction type (cohesion, repulsion, direction, and obstacle repulsion). This is similar to the techniques used by Jung et al. and Saldaña et al. [70, 101].

The resultant vector generated by an agent's interaction with other agents and obstacles is referred to as the agent's *interaction vector*. The resultant vector generated between two agents is referred to as the *inter-agent vector*. The vector applied to an agent to influence its movement towards a destination is referred to as the agent's *destination vector*. The weighted combination of the *destination vector* and the *interaction vector* produces the *movement-direction vector*. *The movement-direction vector* indicates the direction an agent may move.

2.3 Boid-based model

The model introduced in Figure 2.2 is based heavily on the work by Reynolds and other authors on boid-based swarms.

Hereford [55] and Barnes et al. [11] model static swarms using a bi-variable technique. A bi-variable model is based upon inter-agent cohesion and repulsion, which appears as the *interaction vector* above.

Gazi and Passino also used this bi-variable technique to examine inter-agent interactions when creating stable swarm structures and ensuring agents remain part of a swarm while not colliding [38, 39, 40]. They define the degree to which an agent remains cohesive to a swarm as an agent's stability.

If a swarm is to be goal based, the swarm is modelled using the *interaction vector* and a *destination vector* to create the *movement-direction vector* as discussed by Saldaña et al., Stranders et al., Nash and Koenig [101, 147, 102].

The first swarming model to use three components was the Boid model [124]. In the Boid model, cohesion and repulsion are used to produce an *inter-agent vector*. The main difference in the model is how the *destination vector* is introduced. In a Boid swarm

the *destination vector* is not based upon a fixed destination. It is determined by each agent communicating with its neighbours to generate a consensus-based direction. Each agent calculates an average of the neighbours *movement-direction vector* and applies the result as a *destination vector*. This consensus-based movement creates a 'flocking' effect [72, 124]. This cooperative method of creating movement can be seen in the formation of fish shoals as discussed by Yang et al. [160] and Pearce et al. [112]. The same flocking characteristic also occurs in starling murmurations as discussed by Campbell and Samsel [19], and Zhang et al. [164].

Barnes et al., Bennet and McInnes, Cai et al. Correl and Rus, Dinolov et al. and Ekanayake and Pathirana take a different approach to creating a *destination vector* [11, 10, 14, 18, 24, 28, 32]. They generate a *destination vector* in a similar manner to that described in Figure 2.2 using the *interaction vector* and the *destination vector*.

2.4 Swarm cohesion

Several views of cohesion exist within the swarm research community. Cohesion, in some cases, is considered as an agent moving towards the centroid of a swarm. The centroid is the centre of the swarm. This approach is used by Gazi and Passino who measure stability based on changes in distance from the centroid of a swarm [42, 40]. They define stability as the 'degree' to which a swarm will remain a coherent entity. Shinichi et al. [6] also use the concept of the centroid of a swarm to define a metric to measure stability.

Alternatively Long et al. [120], Shinichi et al. [6] and Ekanayake and Pathirana [32] refer to cohesion as an 'attractive force' and define cohesion as being localised to an agent and its 'visible' neighbours. The visibility they discuss is determined by a sensor that provides localised proximity information that includes angles and distances to neighbouring agents.

Similarly, this thesis will view cohesion as the interaction of an agent with its local neighbours. Agents are viewed as being autonomous using only localised proximity information. The Boid model requires information about the swarm's structure, the positions and directions of neighbours. This requires a communications infrastructure. The model in this thesis does not require this information and therefore does not require a communications infrastructure.

This thesis, when analysing the data captured from an experiment, will only use the centroid as a means of tracking the position of a swarm. The centroid and the logic to

identify it will not be used by agent algorithms for coordination.

Cohesion is based on the principle that all agents will remain part of their immediate neighbours' 'cluster' and will 'flock' together in a 'localised' manner [151, 11, 10, 12, 14, 53, 62]. Localised being that the agents will only be 'aware' of their immediate neighbours.

Flocking, in this thesis, should be considered as the process of agents moving towards each other to attain their most stable position [46, 103] which is the centre of mass of their immediate neighbours (Figure 2.2).

The cohesion vector is calculated by summing the relative position vectors identified from the origin agent (b) to each neighbouring agent. This vector is divided by the total number of neighbour agents (Figure 2.2) to produce a resultant cohesion vector. The closer a neighbouring agent is to the agent of interest then the smaller the cohesion vector generated.

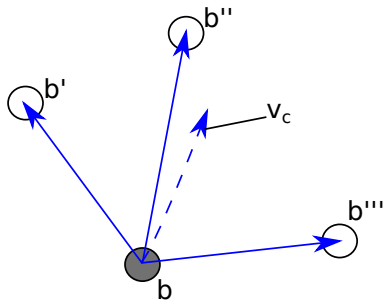


Fig. 2.2: Cohesion: Origin b

Formally the cohesion vector $v_c(b)$ for agent b is the vector calculated by summing the vectors bb' formed from the agent to each of its neighbours $b' \in nbr(b)$ [53] and dividing by the number of neighbours.

A neighbour of b is any agent within the swarm S that is within neighbour range:

$$nbr(b) \triangleq \{b' \in S \ : \ \|bb'\| <= C_b\} \tag{2.1}$$

$$v_c(b) = \frac{\sum_{b' \in nbr(b)} bb'}{|nbr(b)|} \tag{2.2}$$

2.5 Swarm repulsion

Repulsion is defined by Reynolds, Kawabayashi and Chen, and Shinichi et al. as the tendency for an agent to move away from another agent that enters its repulsion field [124, 72, 6]. This creates a 'field effect' around the agent such that when another agent enters that area a vector is applied to prevent the agents colliding. Repulsion is also applied to agents when they interact with obstacles, this is covered in Figure 2.6.

Kawabayashi and Chen [72], Reynolds [124] and Aso et al. implement repulsion as a vector at a boundary with a fixed magnitude (Figure 2.3).

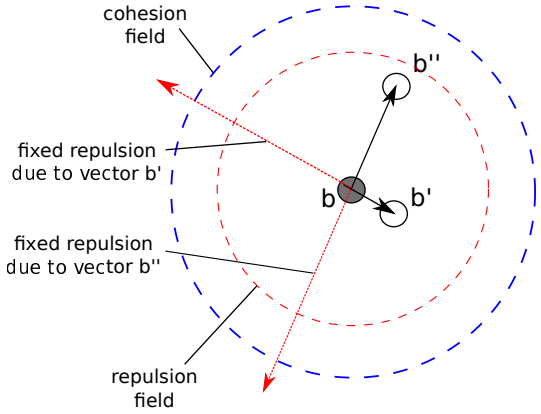


Fig. 2.3: Agent fixed magnitude repulsion

This approach produces a resultant repulsion vector that is based upon the angles at which the neighbour agents approach an agent without considering the proximity of the neighbours to the agent.

In this thesis the repulsion vector has a graduated magnitude. Each neighbouring agent's repulsive effect is applied proportionally (Figure 2.4). When an agent encroaches upon another agent the degree of the field intrusion is mapped to a value in the range $0 \rightarrow 1$.

This affects the magnitude of the repulsive vectors that are applied and therefore the resultant repulsion vector (Figure 2.6).

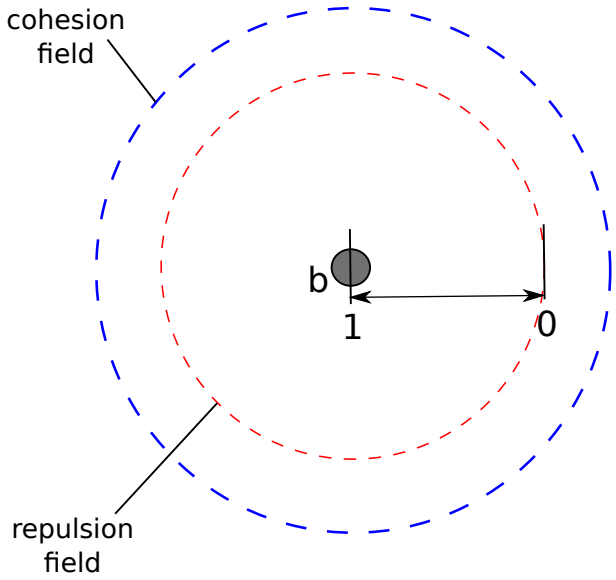


Fig. 2.4: Graduated agent repulsion

This technique changes the repulsion vector such that the direction reduces the probability of a collision. In this thesis the inter-agent repulsion will be calculated as the average of all the proportional repulsion vectors (Figure 2.5).

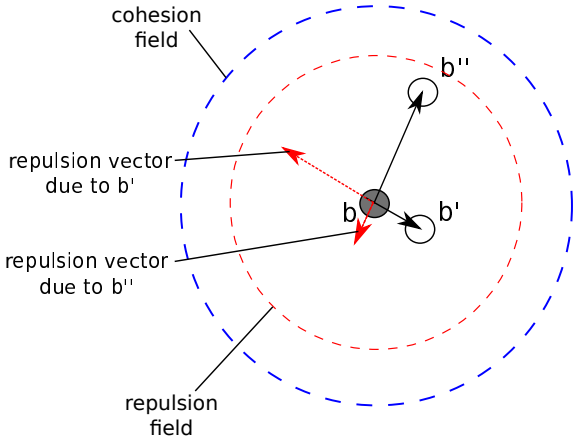


Fig. 2.5: Proportional agent repulsion

Figure 2.6 shows a comparison of the two repulsion models with the proportional repulsion magnitude shown in green and the fixed magnitude shown in red. The two models produce different repulsion angles. The angle produced by the proportional model increases the distance agent (b) will move away from b' when motion is applied. This reduces the chance of a collision between the two agents. The proposed proportional model is therefore suited to swarm's where agent collisions may cause problems.

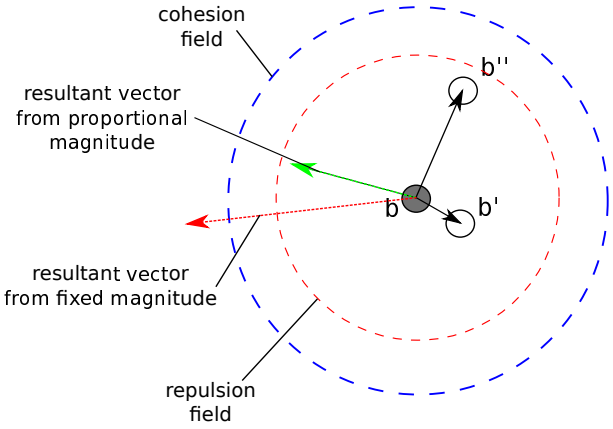


Fig. 2.6: Repulsion comparison

To calculate the total inter-agent repulsion the neighbours that are within the repulsion field must be identified. This is shown in Figure 2.3.

$R(b)$ is the set of all agents that are within the repulsion field. R_b is the repulsion field and $\|bb'\|$ is the distance between b and its neighbour b'.

$$R(b) = \{b' \in S \ : \ \|bb'\| <= R_b\} \tag{2.3}$$

$v_r(b)$ is the repulsion vector generated for agent b based on the proximity of its neighbours. If $R(b)$ is empty then $v_r(b) = 0$ otherwise it is given by equation 2.4. The proportion of field intrusion is calculated by $1 - \frac{\|bb'\|}{R_b}$. The field effect distance R_b is the range around the agent where the repulsion effect is introduced to prevent collisions.

$$v_r(b) = \ - \frac{1}{|R(b)|} \left(\sum_{b' \in R(b)} \left(1 - \frac{\|bb'\|}{R_b} \right) bb' \right) \tag{2.4}$$

2.6 Swarm agents/obstacles interactions

Obstacles, like agents, can be represented as a point in the system. As an agent moves it may enter an obstacle's *obstacle repulsion field* causing the agent to move away.

In this thesis agents are modelled with a fixed obstacle repulsion distance O_b where a repulsion vector is applied. The repulsion is then a vector of magnitude O_b. If more than one obstacle is within the field effect agent the total repulsion vector is the sum of the repulsion vectors due to each obstacle Figure 2.7. The result is normalised and scaled such that the magnitude is the same as the field distance O_b.

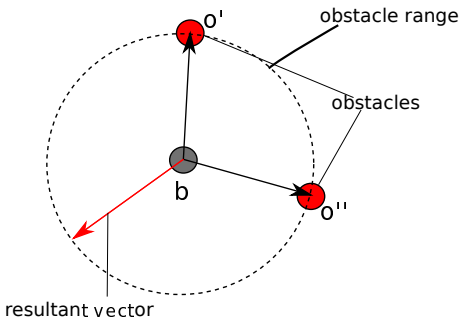


Fig. 2.7: Obstacle repulsion

Equation 2.5 shows the resultant repulsion vector $v_o(b)$ for an agent. $\{o \in O : \|bo\| <= O_b\}$ is the set of obstacles that are within range of agent b. O is the set of obstacles. The obstacles are identified using the distance between an agent and an obstacle $\|bo\|$ and comparing the result to the fixed obstacle repulsion range O_b. The result is calculated by scaling the normalised sum of the normalised vectors $(ob)\hat{}$ by O_b. Note that $\hat{}$ is the equivalent of $\hat{v} = \frac{v}{\|v\|}$ the normalised vector.

$$v_o(b) = O_b \left(\sum_{o \in O \ : \ \|ob\| <= O_b} (ob)\hat{} \right)\hat{} \tag{2.5}$$

2.7 Swarm direction (goal based swarms)

There are two directional aspects to swarm motion. The *interaction vector* which is the vector created by inter-agent reactions through the cohesive and repulsive fields as discussed in Figure 2.4 and Figure 2.5 and the vector for avoidance of obstacles Figure 2.6. The *destination vector* is applied to influence the motion of a agent towards a particular coordinate [15] and the *interaction vector* to maintain the swarm's structure. This model is used by Barnes et al. [11, 10], Bennet and McInnes [14], Cai et al. [18], Correll and Rus [24], Dinolov et al. [28] and Ekanayake et al. [32]. This thesis uses a similar technique, defining a single destination as a *destination vector* for goal based swarms. The application and effect of multiple destinations is discussed in future work.

$v_d(b)$ (Equation 2.6) is the *destination vector* where d is the destination.

$$v_d(b) = bd \tag{2.6}$$

2.8 Weighted movement-direction model

An agent's *movement-destination vector* is the sum of all the component vectors (v_c, v_r, v_d, v_o) (Equation 2.7) [53]. For a vector to be used for movement it must have a magnitude of 1 before the agent's speed can be applied (Section 2.7).

$$v(b) = v_c(b) + v_r(b) + v_d(b) + v_o(b) \tag{2.7}$$

This model is extended by adding a weighting to each of the component vectors. The addition of the weightings allows the influence of each component vector set to be adjusted to produce a bespoke movement vector (§ 2.8). The resultant vector is normalised to produce a unit *movement-direction vector* that can be used to create motion [72]. The agent's speed characteristic is used along with time (t) [35, 41] to determine an agent's next position. This derived vector is the *movement vector*.

The purpose of a weighted aggregation model is to alter the level of influence of each component of the equation. This technique is generally referred to as a *'weighted sum*

aggregation' or *'ordered weighted averaging'*. The technique is applied to optimisation algorithms such as PSO (Particle Swarm Optimisation) and involves applying all the possible combinations of weightings to a multi-variable expression to obtain an optimum output [87, 158].

In this thesis the technique of *weighted sum aggregation* is applied to the vector calculations to allow tuning of the swarming algorithm of an agent and to change the degree of influence to obtain a required swarming effect.

The tuning is applied to each component as a weighing factor k Equation 2.8. The weightings (k_c, k_r, k_d, k_o) are applied before normalising the *movement-directional vector*. This change of bias allows levels of importance to be applied to a system characteristic i.e. $k_c > k_d$ implies it is more important for the agents to remain together than it is to travel towards the destination. This technique is similar to those identified by Muniganti and Pujol in their survey of mathematical modelling techniques [100].

Weightings can be applied in several ways. The weighting can be applied as a set of arbitrary integer values (12, 67, 99) or as a set of values that always have a summed value of 1 e.g. 0.5, 0.25, 0.25. Either of these techniques are acceptable as the resultant vector is normalised following the application of the weighting. This thesis implements the weightings as a set of arbitrary integer values (Equation 2.8). Where k_c is the weighting factor for cohesion, k_r is the weighting factor for repulsion, k_o is the weighting factor for obstacles and k_d is the weighting factor for a destination.

$$v(b) = k_c v_c(b) + k_r v_r(b) + k_o v_o(b) + k_d v_d(b) \tag{2.8}$$

Special cases of Equation 2.8 can be applied to a swarm model. A swarm with no destination can be modelled with the destination weighting set to zero to create the model shown in Equation 2.9 as used in Chapter 7. This is also known as the *interaction vector*

$$v(b) = k_c v_c(b) + k_r v_r(b) + k_o v_o(b) \tag{2.9}$$

A swarm that does not interact with obstacles and has no goal (destination) can have

k_o and k_d set to zero creating the model as shown in Equation 2.10. This is also known as the *inter-agent vector* as discussed in § 3.7 on page 40.

$$v(b) = k_c v_c(b) + k_r v_r(b) \tag{2.10}$$

Equation 2.10 is also the model used in the calculation of the swarm magnitude metric as discussed in chapter 3 where $v(b) \equiv P(b)$.

2.9 Modelling movement

Each agent within a swarm calculates its *movement-direction vector* based on its *interaction* and *destination* vectors. The *movement vector* (b_{pos}) is calculated using the unit *movement-direction vector* of Equation 2.8 multiplied by the time elapsed (t) in the system and the speed characteristic of the agent (s_b).

This process is carried out for every agent in the swarm to create the entire swarm's next position.

$$b_{pos} = s_b t\big(v(b)\big)\hat{} \tag{2.11}$$

The increment in the location of agent b over time interval t is shown in Equation 2.11 where s_b is the speed of agent b. Models of time are discussed in § 2.13.

2.10 Stable swarm structures

A swarming behaviour can be created using only cohesion and repulsion. This technique is known as a bi-variable model [11, 10]. The bi-variable model produces natural geometric structures. The structures tend to be based on equilateral triangles and when the distribution of the agents allows, regular hexagons are formed. These structures only occur when the repulsion and cohesion field effects produce a distribution such that

an agent's detected neighbours do not extend beyond the first agent detected in any direction [109, 110]. The effect of field effect ranges on a swarm's structure is discussed in chapter 4.

The most stable state for agents is for all agents to be equidistant with equal angles. If two agents are in close proximity they will naturally adhere to each other due to the proximity rule (cohesion) (Figure 2.8); repulsion will ensure a minimum distance is preserved. In the case of 3 agents a triangle will form. In the case of 4 agents the most stable shape will be a diamond with the centre agents joined. With 5 and 6 agents a triangular lattice will emerge and with 7 agents a stable hexagon will form. The hexagon (Figure 2.9) is the most stable structure with all agents being equidistant and all angles between each neighbouring agent equal [11, 41]. These structures are seen throughout the natural world [123].

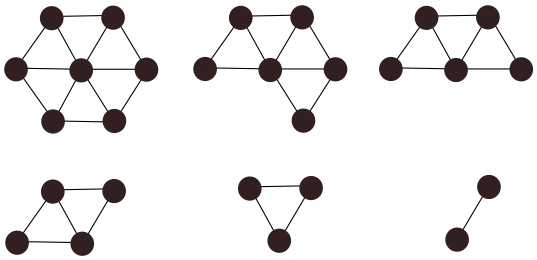


Fig. 2.8: Stable swarm formations

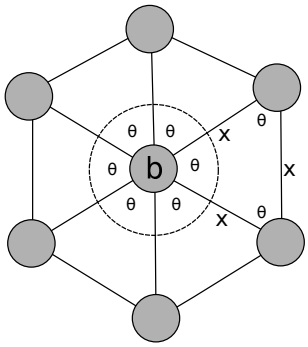


Fig. 2.9: Stable hexagonal formation

2.11 Resultant swarm model

The swarm model created by Equation 2.8 with suitable weightings will allow a swarm to form 'stable' structures such that the agents will remain connected (Figure 2.11) and over time migrate to an optimum overall structure for the models parameters. The parameters are the field effect ranges, the cohesion and repulsion magnitude models and the weightings.

The initial random deployment of a set of agents to create a swarm produces a 'disorganised' state. The disorganisation is caused by the varying cohesion and repulsion vectors that are generated by the inter-agent relationships. Following the initial deployment the magnitudes will create movements that gradually stabilise the swarm structure to a level of movement that best fits the model parameters [113, 155]. The point of equilibrium for the swarm and the resultant structure is dependent upon the agent's cohesion and repulsion fields level of overlapping. This is discussed in § 4.2 and § 4.3.

When modelling swarms it is common practice to have the agents in constant motion [83, 49]. In this thesis agents are modelled moving at a constant speed with no inertial effect such that an agent can move freely within the system plane. The only exception to this will be if an equilibrium state is encountered where the summed vectors produce a null vector. If this occurs the agent will stop moving.

2.12 Swarm deployment

Using the methods discussed in this chapter, a swarming behaviour emerges from a collection of agents. The initial deployment of a swarm may be a random dispersal of agents such that the swarm is in a disorganised state (2.10), caused by an instability in the magnitudes that are acting upon each of the agents (as detailed above). Based upon the application of the models discussed, the swarm will initially move in such a way as to balance all the vectors, resulting in a period of disorganisation where the swarm's movement towards a goal is limited, as the vectors generated to disperse the agents outweigh the directional vector.

This phase of the swarm's life cycle is the 'initialisation phase' (Figure 2.12). When the initialisation phase is over, the vectors (cohesion, repulsion, and direction) become more balanced and the swarm forms a more regular shape, such as a hexagonal lattice, where all the angles and lengths (distances between agents) tend toward being

equal (Figure 2.11)

The effects can be seem in the screenshots (Figures 2.10, 2.11, 2.12) from the simulator discussed in § 2.13.

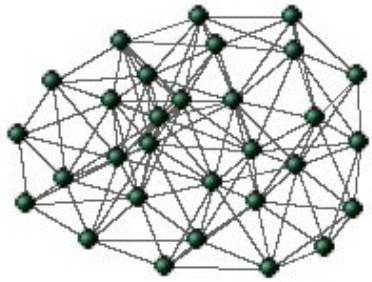


Fig. 2.10: Disorganised swarm

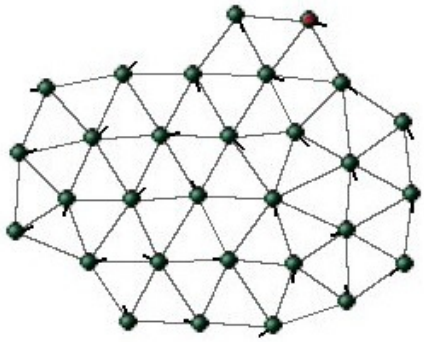


Fig. 2.11: Stable swarm

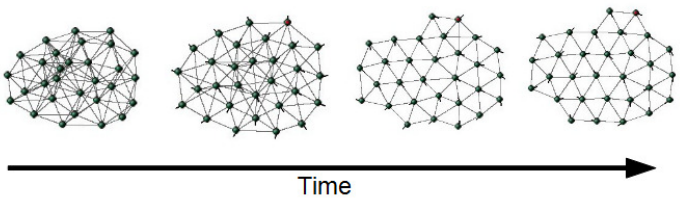


Fig. 2.12: Swarm stabilisation

2.13 Swarm simulator

Swarm behaviours can be investigated by means of experiments with physical robots or by means of simulations. The latter approach has the advantages of scalability, generalisation, speed of development, and cost. This thesis is based upon data generated from simulated swarms. There are several open source robotic simulators available, the most popular being ARGoS, Player/Stage, and Gazebo.

ARGoS is described as a multi-physics simulator and has gained interest in the swarm robots community. In 2011 Luca and Caro published an overview of the simulator's engine discussing how the system functioned [116] and the philosophy behind its structure. They also published a framework for using the simulator in 2012 [75].

Player/Stage and Player/Gazebo are used in many projects including projects simulating single robots as discussed by Song and Gupta [142] and also multiple-robot swarming simulations as described by Lei et al [80]. There are projects that have simulated the use of pheromone trails when simulating foraging based swarms as discussed by Shi et al [139]. Shi et al. have also published an overview of the scenarios in which Player/Stage can be used [140].

The Webot [86] simulator, which is a commercial product, has been used successfully in other research projects such as the swarm simulations developed by Srivastava and Nandi [143]. One problem found with the product was that it was restrictive in terms of how much of the system could be configured to meet the needs of the thesis. Another factor that had to be taken into consideration was the high cost of a licence for the PRO version of Webot.

These simulators all provide a discrete time simulation environment. The main purpose of these simulators is to visualise either an individual robot or a swarm of robots based on a model that is defined through bespoke libraries and configuration parameters. On the other hand, in this thesis the main purpose of each simulation is to log all the positional and vector data associated with every agent at each discrete time interval. Due to this disparity in approaches it was decided to develop a simulator whose main purpose is the collection of data on distance, positions, distribution and inter-object vector magnitude influence.

This section discusses the design, development and usage of the simulator used in this thesis and the creation of the raw experimental data. The section also discusses how the data is processed to produce the aggregated data required for visualisation.

2.13.1 Simulator overview

The simulator has two distinct components: a graphical design/simulation tool (Appendix B.1) and a command line based simulation-only tool (Appendix B.2). Both parts of the simulator are written in Python 3 [119] using an object model as shown in appendix E. Both use the same modelling engine by sharing the base classes. The final object model is similar to that proposed by Vankerkom and Yu for swarm visualisation [152]. The simulator design is also influenced by the *'main loop'* proposed in the ARGoS simulator [116].

2.13.2 Simulator architecture

The main purpose of the graphical environment is for the setup of an experiment's initial configuration. This is achieved by positioning the agents, destinations, and obstacles in an environment and saving the configuration as a simulation file. As a secondary purpose the graphical environment is capable of running small scale simulations. The command line tool is used to execute the simulation experiments designed using the graphical tool.

The graphical tool, shown as (1) in Figure 2.13, uses PyGame [136] as its graphical presentation layer. PyGame supports several rendering engines; in this application the default SDL rendering engine is used. The graphical simulator runs in real-time and is capable of simulating small swarms of < 150 agents on a PC with an `Intel Core i7-4770 CPU @ 3.40GHz * 8` processor. This swarm size limitation is due to the Python code being executed on a single processor core. There is also a limitation in the performance of the graphical engine due to the rendering being performed by the interpreter.

The command line tool, shown as (2) in Figure 2.13, reads in the experiment configuration file generated via the graphical tool, shown as (1). The command line tool uses simulated discrete time (Figure 2.13.3) and is able to run with arbitrary sized swarms without real-time processing limitations. The command line tool simulates the swarm and generates the initial data extract (3a). The data extract is then loaded into a MySQL database (3b) and the data is then aggregated to create the complete dataset for the experiment (4). The processes 5 and 6 are discussed in Figure E.7. This thesis deals with arbitrary sized swarms, so simulations are designed in the graphical environment but executed using the command-line-based simulator.

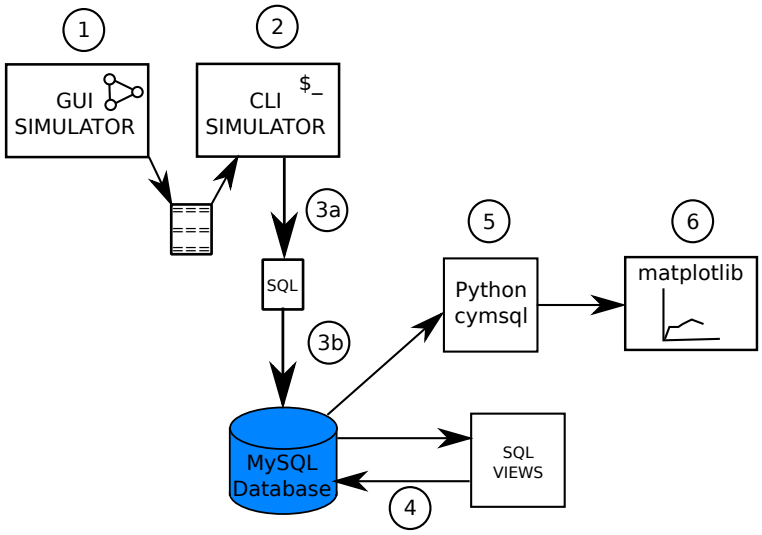


Fig. 2.13: Simulator process overview

Figure 2.13 shows the stages of the simulation from developing an experiment (1) through to the production of simulation results (6).

2.13.3 Simulated time

There are two options for representing time when modelling a swarm: continuous time and discrete time. Continuous time [52] is *dense*: between any two points in time there is another point. Discrete time [35, 41, 128, 55, 100, 110] on the other hand proceeds in 'ticks' with no intermediate time points. In this thesis discrete time is used. This same approach is identified by Muniganti and Pujol in their survey of mathematical swarming models [100].

Vision based coordination for robots was a subject of great interest in the 1980s and 90s [27]. This interest moved to omni-directional cameras as a means of determine position and mapping through image analysis in a process known as SLAM (Simultaneous Localisation And Mapping) [149, 142]. A general purpose omni-directional camera can operate at speeds between 1Hz and 60Hz, depending upon the resolution of the images and the accuracy of the positional data required.

For the identification of an agent's position, a GPS with a sample rate in the same range may be used. For example, the SparkFun Venus GPS [33] operates at up to 20Hz.

The simulator allows the sampling rate to be adjusted to provide a model that is as close as possible to the physical sensors. For the experiments in this thesis, the sample rate is set at 10Hz, clearly within the scope of currently available sensors. This gives a 'tick' interval of 100ms.

2.13.4 Simulated field effects

The ranges for the cohesion, repulsion, and obstacle avoidance fields are user-configurable parameters in the simulator, as is the location of a destination goal, if present. The simulation of the operations of cohesion, repulsion, obstacle avoidance and goal-seeking then directly follow the definitions given by equations 2.2, 2.4, 2.5, and 2.6. The details of the implementation are shown in Appendix E.1.

2.13.5 Simulated agent movement

The motion model of the simulation is implemented through the modelling of vectors that influence an agent's resultant direction. The vectors that model the swarm environment are the cohesion and repulsion vectors created by inter-agent and inter-object interactions.

Agent positions are modelled using floating point numbers. These coordinates are translated to integer based (x, y) co-ordinates for the presentation layer. The integer translation is only for the visualisation of the swarm. This is the same approach used by Vankerkom and Yu in their paper on swarm visualisation [152]. They model the agent using a class that consists of positional variables of type double. This is also seen in the SwarmVis software developed by Miner and Kasch [94].

The incremental positions of the agents are calculated based upon the simulated time slice, 'tick', as discussed in § 2.13.3, and the agent speed, given as a parameter of the simulation. These parameters are used as t and s_b, respectively, in applying Equation 2.11 to the calculation of the position of each agent at the next 'tick'. This movement is implemented within the simulator as shown in Appendix E.1.2.3.

2.13.6 Simulator data capture

To enable the analysis of a simulation run, the simulator generates an SQL database. As a simulation executes, at each tick, the state of each agent in the swarm is captured

and the data is saved as a pair of SQL insertions, (*tick number*, *agents' states*), that are added to a set of transaction files. The implementation of the data capture component of the simulator is described in detail in Appendix E.3. This approach generates a large amount of data but allows for very detailed, offline analysis of swarm behaviour.

2.14 Conclusion

This chapter notes the trend in using vectors as a modelling technique for swarms and discusses the use of field effects in determining agent movement. The chapter then introduces the mathematical model that is applied throughout the thesis. The introduction covers the cohesion model that ensures agents remain part of a swarm and the repulsion model that ensures agents do not collide with each other, thus maintaining a stable swarm structure. The chapter also introduces two additional aspects of swarming to the model: goal-based direction and obstacle avoidance. Finally, the chapter discusses the simulation of swarms. All simulations in this thesis are carried out using the simulator described in this chapter. The data created by each simulation is aggregated to generate the final datasets that allow the characteristics of the simulated swarm to be evaluated. Data analysis results are visualised from the aggregated data.

3. SWARM MOVEMENT METRIC

This chapter examines the distance metric as a mechanism to measure the internal movement of agents and introduces a new *magnitude based metric*. The internal movement of a swarm is identified by analysing the changes in the inter-agent interactions. The two metrics differ in their approach to identifying the changes. The distance metric uses variations in the inter-agent spaces, as used by Navarro et al. [104]. The new metric, devised as part of this thesis, uses the magnitudes from the agents' *inter-agent vectors* that are induced by agents' field effects as defined in Equation 2.10.

Both metrics allow a comparison of the effects of different swarming algorithms on a swarm's structure. The type of information that can be derived from each of the metrics is compared in § 3.12.

The magnitude based metric is used in chapter 5 to identify the effects of different coordination algorithms. In chapter 6 the metric is used to identify the effects of both obstacles on a swarm's movement and the encapsulating behaviour a swarm exhibits when using *concave reduction*.

3.1 Inter-agent vector magnitude effect on internal movement

Figure 3.1 shows the cohesion and repulsion vector contributions to $v_c(b)$, $v_r(b)$ due to neighbour b', as given in equations 2.2, 2.4. Notice that the vectors are along the line of separation bb'.

Using the cohesion and repulsion vectors generated by the relationship of b' to b a resultant vector can be calculated. This vector creates an agent characteristic that can be used as a metric. Summing the vectors creates a resultant vector with a magnitude that affects the agent. Summing the vectors also provides an indication of the direction an agent will move based on the relationship. This is defined in chapter 2 as the *inter-*

agent vector.

From here throughout § 3.2 and § 3.3 imagine agent b has just a single neighbour b' and consider the effect of b' on $v(b)$, the *inter-agent vector* of b.

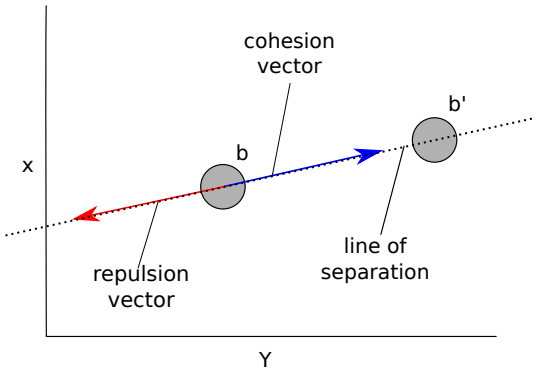


Fig. 3.1: Vectors on line of separation

3.2 Swarm movement analysis

The repulsive and cohesive vectors are generated for an agent through the intersection of their field effects (§ 2.4 and § 2.5). There are a limited number of intersections that can occur; These are illustrated in Figures 3.2, 3.3, 3.4, 3.5.

Figures 3.2, 3.3, 3.4, 3.5 show the cohesion of an agent pair as $k_c v_c$ and the repulsion as $k_r v_r$. The example data extracts (Tables 3.2, 3.3, 3.4, 3.5) are generated from the simulator using the parameters in table 3.1 that create a basic swarming behaviour. The tables show the simulation results. The simulation consists of 200 agents over a 20 second period. The simulation produces a neighbour extract of 248,798 records.

Weight Component	**Swarm**	**Description**
Sample Rate	100	ms - Unit sampling interval
k_c	5	weight adjuster for cohesion bias
k_r	15	weight adjuster for repulsion bias
k_d	0	weight adjuster for directional bias 0 for static baseline 100 from directional
Repulsion field	70	units
Cohesion field	80	units
Speed	20	units/s

Tab. 3.1: Swarm parameters model

Figure 3.2 shows two agents within each others cohesion fields but sufficiently distant to be outside of the repulsion fields. The 'neighbour region' and 'repulsion region' are the limits of the field effects for cohesion and repulsion. In this case $k_c v_c > 0$ and $k_r v_r = 0$: the result is the agent's resultant magnitudes cause the agents to move towards each other. Table 3.2 shows the repulsion magnitude with a value of 0. The only influence on the agent pairs are cohesive vectors.

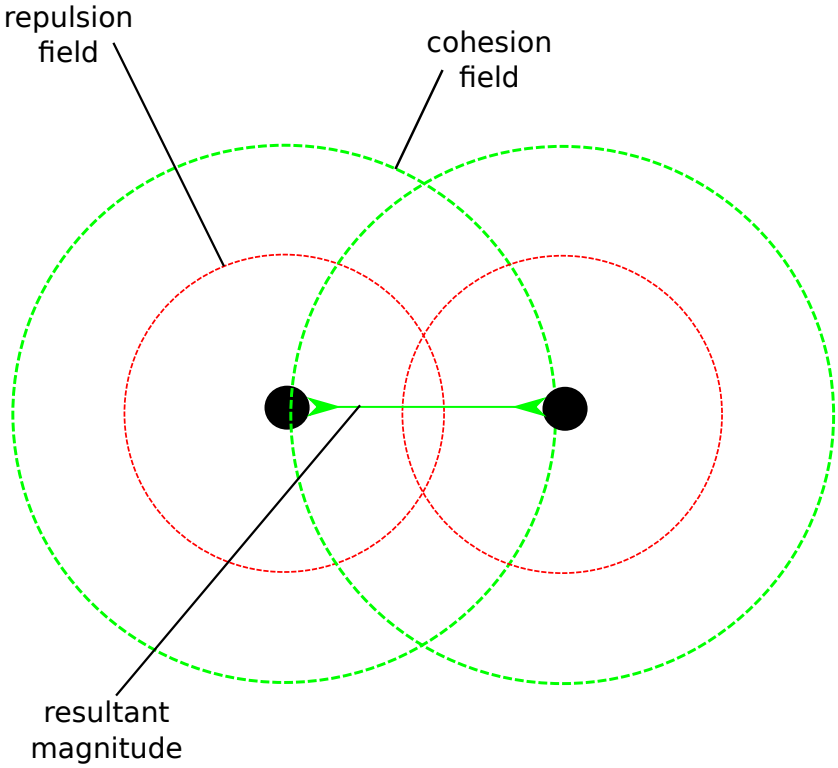


Fig. 3.2: Internal movement cohesion (no repulsion)

In tables 3.2, 3.3, 3.4 and 3.5, **Log** is the sample identifier, **Id** is the unique identifier for an agent and **N.Id** is the Id of the agent neighbour.

Log	Id	N.Id	Distance	Cohesion	Repulsion
0	1	3	70.50359957272653	352.5179978636327	0
0	1	100	71.78005530038806	358.9002765019403	0
0	1	151	78.33995887998715	391.69979439993574	0
0	2	99	72.04066804327307	360.20334021636535	0

Tab. 3.2: Data extract ($k_r v_r = 0$)

Figure 3.3 shows two agents close together with repulsion dominating cohesion such that $k_c v_c < k_r v_r$. The resultant vector will direct the agents away from each other. Table 3.3 shows the repulsion magnitude with a value greater than cohesion.

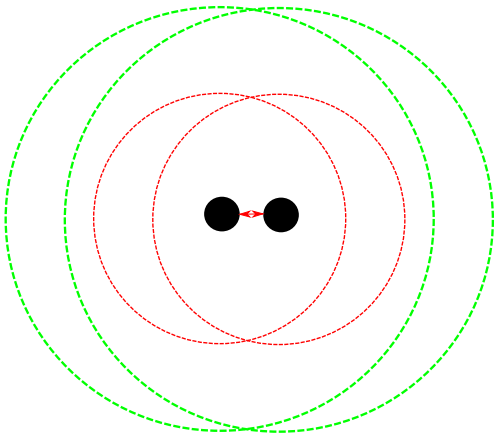


Fig. 3.3: Internal movement repulsion

Log	Id	N.Id	Distance	Cohesion	Repulsion
0	1	2	28.325225929649267	141.62612964824635	1544.860149837827
0	1	6	41.48517221724064	207.42586108620318	721.7173648240145
0	1	7	35.264128136470426	176.32064068235212	1034.271010913942
0	1	8	43.545037655009644	217.72518827504823	637.9075999959364

Tab. 3.3: Data extract ($|k_c v_c| < |k_r v_r|$)

Figure 3.4 shows two agents close together but with cohesion vector magnitudes greater than the repulsion magnitudes $|k_c v_c| > |k_r v_r|$. The resultant vector will draw the agents together. The magnitude of the resultant cohesion vector will be reduced due to the cancelling effect of the repulsion vector. Table 3.4 shows a data extract with the cohesion magnitude greater than repulsion.

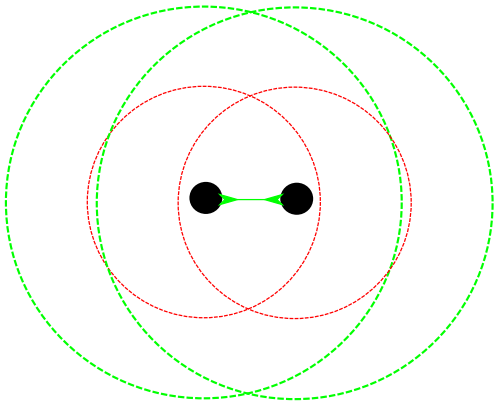


Fig. 3.4: Internal movement cohesion

Log	Id	N.Id	Distance	Cohesion	Repulsion
0	1	5	64.17214469587854	320.86072347939273	95.35676418993891
0	1	9	63.880497718571355	319.4024885928568	100.58590062663305
0	1	95	65.6152270119206	328.07613505960296	70.16681717258929
0	1	152	63.10700566424517	315.53502832122587	114.68844031437281

Tab. 3.4: Data extract ($|k_c v_c| > |k_r v_r|$)

Figure 3.5 shows two agents close together with $|k_c v_c| = |k_r v_r|$ the resultant vector will be a *null vector* and the agents will have no influence upon each other due to the magnitude of the resultant vector being zero. Table 3.5 is an extract from the `NEIGHBOURS` table. The data shows an extract that is near equilibrium. The simulation produced no null magnitude results.

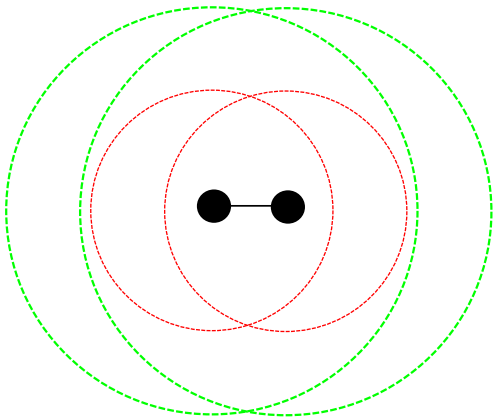


Fig. 3.5: Internal movement equilibrium

Log	Id	N.Id	Distance	Cohesion	Repulsion
7	76	91	55.390312278311875	276.9515613915594	276.9468428106153
24	75	6	55.39032191143417	276.95160955717085	276.9466120367043
32	72	38	55.39002603773678	276.9501301886839	276.95370011064875
35	63	64	55.390227283173054	276.9511364158653	276.9488789826377

Tab. 3.5: Data extract ($|k_c v_c| \approx |k_r v_r|$)

3.3 Internal movement and the null vector

When the two vectors (cohesion and repulsion) have magnitudes that are equal and opposite they produce a null vector. This indicates that two agents are optimally spaced for a given set of conditions. Although the agents are at an optimum position it does not mean the swarm is optimally distributed. If a swarm is in a confined space it is possible for an optimum position to be created where the vector magnitude is positive due to a compression effect. This phenomenon is used in the identification of the emergent behaviour of area flooding, covered in chapter 7.

If we consider the equilibrium state (Figure 3.5) the resultant vector of b is $(0, 0)$. A null vector cannot be normalised to produce a directional vector ($\hat{v} = \frac{v}{|v|}$ if $v \neq 0$; 0 if $v = 0$). The effect of the resultant magnitude being a null vector is that the agent will remain stationary. If all agent pairs are in this condition the swarm will stop moving (Figure 3.6).

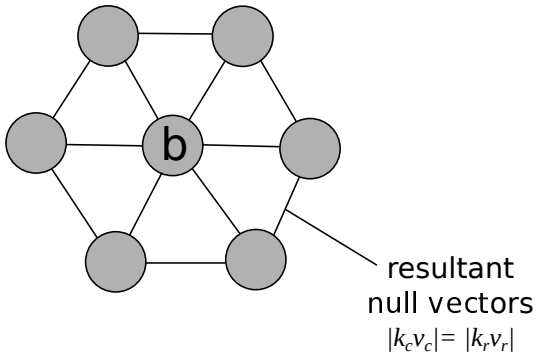
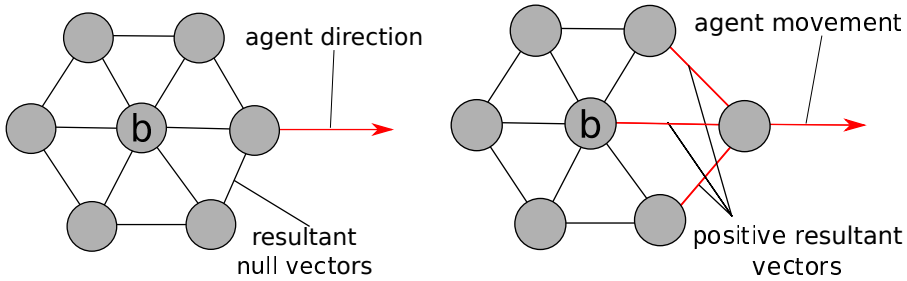


Fig. 3.6: Equilibrium with null vectors

Due to the independent nature of the agents this situation is very rare. The residual motion that persists in a swarm is the background 'noise' or 'jitter' that an algorithm creates.

If a swarm is goal-based the additional *directional vector* will prevent all agents simultaneously producing null vectors (Figure 3.7).



(a) (t) (b) $(t+1)$

Fig. 3.7: Directional movement and the null vector

3.4 Residual internal movement (Jitter)

Due to the dynamic nature of a swarm maintaining optimum internal movement as in (Figure 3.5) a stationary swarm is highly unlikely. The agent pairs will fluctuate between the 3 states (Figures 3.3, 3.4, 3.5). This alternation between the states is *jitter*. The degree to which this variation occurs can be measured using either the change in distance between the agent pairs, or the change in the resultant magnitude

between the agent pairs. Jitter is motion that is produced to maintain the structure of a swarm. A coordination algorithm that produces minimal jitter is generally more desirable. Jitter (the fluctuation in and between the states) is an indication of the efficiency of an algorithm and an integral component of a swarm's measurable behaviour.

3.5 Magnitude based metric

Magnitude based internal movement (*agent resultant magnitude*) is measured by identifying the balance between the repulsion and cohesion between agents. 'Jitter' in the case of the *agent resultant magnitude* metric is measured as the variance of the potentials created by the agents. The identification of this variance produces the clarifying part of the *agent resultant magnitude* metric. The *agent resultant magnitude* is identified by Gazi and Passino [42] and Barnes et al [12] as a 'resultant characteristic' of a swarm. There are two ways of using the cohesion and repulsion in identifying a resultant vector. The two vectors can be added as absolute values to give an overall 'size' to the magnitude that is affecting each relationship. Alternatively the resultant magnitude can be the sum of the actual magnitudes. The repulsion vector has a negative magnitude and the cohesion vector has a positive magnitude. In this thesis the magnitude analysis will be based on summing the two actual vectors to determine the result of the inter-agent interaction. This thesis will refer to the resultant magnitude as the 'agent resultant magnitude' of the relationship. The 'state' of a swarm is the effect the environmental constraints and algorithms have upon the agent resultant magnitude. It is a part of the 'quality' measure for a swarm's performance.

If the *agent resultant magnitude* is a negative value (absolute values would prevent this analysis) the swarm's bias is to expand. This is seen in the disorganised stage of a swarm. If the *agent resultant magnitude* is positive then the swarm is exhibiting a tendency to contract and this indicates the swarm is a cohesive entity. This could also be described as the swarm being 'sticky' as the agents bias is to 'pull' towards each other.

The *agent resultant magnitude* on its own does not give a complete measure of a swarm's internal state. There needs to be a qualifying component to the metric that identifies the degree of deviation in the resultant magnitude, this is the *jitter*. The smaller the degree of deviation the more uniform the structure of the swarm. These two components identify the degree to which a swarm has progressed towards a stable state.

The *agent resultant magnitude* provides a view of the swarm's state through the balance

between the repulsive and the cohesive vectors that are being applied to each agent. The variance component identifies the degree to which the swarm has stabilised. The ideal status for inter-agent interactions would be for the agents to have a resultant vector (*agent resultant magnitude*) of zero or above. This would indicate that the agents are distributed such that they are at their distribution limit (outer most range of the cohesion field) or at a level that causes the agents to 'pull' together. The ideal degree of deviation is zero as this indicates an even distribution of agents. Therefore for a fuller indication of a swarm's *state* both measurements need to be combined. The deviation from the mean clarifying the internal movement and the *agent resultant magnitude* providing an indication of the 'compression' that a swarm is logically experiencing (cohesiveness). These two aspects of a swarm's features are not considered by Gazi and Passino [42] or Barnes et al [12] as a means of quantifying the structure of a swarm in terms of stability.

3.6 Distance based metric

The distance based metric considers the effect of the resultant vectors upon a swarm in terms of how the agents are physically distributed: i.e. only the inter-agent distances and the deviation from the mean of the agents (jitter) are considered. As with the *agent resultant magnitude* metric the variations are important to determine the agent distribution. The standard deviation from the mean allows the internal 'characteristic' of the measure to be realised. If the standard deviation is zero then all the agents are evenly spaced. The distance metric does not take into consideration the vector magnitudes between the agents as discussed above. The metric therefore is unable to identify the potential state of the swarm in terms of its cohesive or repulsive state.

Navarro and Fernando describe a mean distance error metric that is based on the variations in distances between inter-agent spaces [104]. This is the same as the standard deviation of the distance based internal movement metric as described here.

3.7 Magnitude based internal movement model

Using the formulae for the calculation of cohesion (Equation 2.2, page 16) and repulsion (Equation 2.4, page 18) for every agent and its neighbours it is possible to calculate an *agent resultant magnitude* value (sum of agent resultant magnitudes). This value represents the overall potential of an agent. This magnitude when normalised produces

a component of the *movement-destination vector* (Equation 2.7) for a swarm. If the agent resultant magnitude is zero (null vector) then the agent will not move. $P(b)$ is the *inter-agent resultant magnitude vector* for agent b defined by:

$$P(b) = k_c v_c(b) + k_r v_r(b) \tag{3.1}$$

Although it is possible for agent b to have a resultant vector of null there could still be a variation in the constituent components. The variation calculation (standard deviation) is shown in § 3.8. Equation 3.2 is the mean of the *agent resultant magnitudes* for an agent and its neighbours where $|nbr(b)|$ is the number of neighbours.

$$\mu_p(b) = \frac{P(b)}{|nbr(b)|} \tag{3.2}$$

To identify the swarm based *agent resultant magnitude* Equation 3.2 must be extended to iterate over all the agents in the swarm. Equation 3.3 shows $\mu_p(S)$ as the swarm based magnitude where the swarm iteration is shown as $\sum_{b \in S}$ and $\sum_{b \in S} |nbr(b)|$ calculates the total number of inter-agent relationships.

$$\mu_p(S) = \frac{\sum_{b \in S} P(b)}{\sum_{b \in S} |nbr(b)|} \tag{3.3}$$

3.8 Variance in agent resultant magnitude metric

The mechanism just described provides an overall indication of the internal movement based on inter agent vectors that produce the *agent resultant magnitude*. This model however is not sufficient to give an indication of the swarm 'state' as an overall metric. To improve the metric clarification is required in terms of the deviation from the *agent resultant magnitude* norm. The variation in the metric is the standard deviation of the entire swarm from the mean of the inter-agent potential magnitudes (Equation 3.2).

The standard deviation is calculated as Equation 3.4 where $\sigma_p(S)$ is the standard deviation at a time t and $\mu_p(S)$ is the mean at the same point in time. $\sum_{b \in S} \sum_{b' \in nbr(b)}$ iterates over every agent in the swarm and its neighbours and $\sum_{b \in S} |nbr(b)|$ calculates the total number of inter-agent relationships.

$$\sigma_p(S) = \sqrt{\frac{\sum_{b \in S} \sum_{b' \in nbr(b)} \Big(P(b') - \mu_p(S)\Big)^2}{\sum_{b \in S} |nbr(b)|}} \tag{3.4}$$

The metric for the internal movement is a set of numbers, the mean and standard deviation of the swarm's internal *agent resultant magnitude* derived from each agent and its neighbour interactions Equation (3.5). The pair $\mu_p(S)$, $\sigma_p(S)$ may be written informally as:

$$\psi_p = \mu_p(S) \pm \sigma_p(S) \tag{3.5}$$

3.9 Distance metric

The distance based internal movement is measured by identifying the mean length of the vectors between an agent and its neighbours. As with the *agent resultant magnitude* a coordination algorithm produces 'jitter' which is the variations from the mean. In the case of the distance based metric the jitter is identified by the changes in the distances rather than the changes in vector magnitude (*agent resultant magnitude*). The distance metric is the mean and the standard deviation 'jitter' of the inter-agent distances.

3.10 Calculating distance based internal movement

The relative position vector generated for an agent b to its neighbour b', bb', is shown in (Equation 2.2). The magnitude of that vector gives the distance between two agents. For an individual agent the average magnitude $\mu_d(b)$ is calculated as Equation 3.6 where b is the agent and $|nbr(b)|$ is the number of neighbours.

$$\mu_d(b) = \frac{\sum_{b' \in nbr(b)} \|bb'\|}{|nbr(b)|} \tag{3.6}$$

Equation 3.6 identifies the mean distance for an individual agent. The mean distance for a swarm is calculated by Equation 3.7. All the inter-agent interactions must be included

for the swarm (S). $\sum_{b \in S} |nbr(b)|$ calculates how many inter-agent relationships exist in the swarm and $\sum_{b' \in nbr(b)} \|bb'\|$ calculates the total distance between each agent and its neighbours. $\sum_{b \in S}$ iterates over all the agents in the swarm (S).

$$\mu_d(S) = \frac{\sum_{b \in S} \sum_{b' \in nbr(b)} \|bb'\|}{\sum_{b \in S} |nbr(b)|} \tag{3.7}$$

3.11 Variance in distance metric

The mechanism above provides an overall indication of the distribution of the agents. This model, as with the agent resultant magnitude model, is not sufficient to give an indication of the internal distribution of the agents. The addition of the standard deviation from the norm clarifies the distribution within the swarm as shown in equation 3.8. $(\|bb'\| - \mu(S))^2$ is the square of the difference in a distance to the mean and $\sum_{b \in S} \sum_{b' \in nbr(b)}$ calculates the number of inter-agent interactions.

$$\sigma_d(S) = \sqrt{\frac{\sum_{b \in S} \sum_{b' \in nbr(b)} \Big(\|bb'\| - \mu_d(S)\Big)^2}{\sum_{b \in S} |nbr(b)|}} \tag{3.8}$$

The distance metric for the internal distribution of the agents is the pair consisting of $\mu_d(S)$, $\sigma_d(S)$ the mean and the standard deviation of the swarm's internal resultant distances from every agent in the swarm. This can be written informally as:

$$\psi_d = \mu_d(S) \pm \sigma_d(S) \tag{3.9}$$

3.12 Conclusion - metric comparison

The two metrics appear to be similar in terms of the measurement of the structure of a swarm. The main difference is in how these two metrics can be used when examining the state of the swarm.

Both metrics identify the state of a swarm with respect to variations in the disbursement of the agents from an average distribution.

The main difference in the metrics is that the distance metric is based upon the physical *distribution* of the agents and the magnitude based metric is based upon the logical *interaction* of the agents.

The distance based metric provides and analysis of the actual distribution of the agents at a point in time and allows the agitation of the swarm to be assessed without considering the possible distribution of agents that the field effects *could* produce.

The *agent resultant magnitude* metric provides a view of the interaction magnitude. This provides an indication of the swarm's potential movement. This is independent of the physical distribution. The lack of dependence on the physical distribution allows the metric to be used in heterogeneous field effect swarms § 8.2.1 where the physical distribution may vary.

Combining the two metrics allows a deeper evaluation of a swarm to be made. Consider the following: the repulsion field is increased but the internal distances do not change as a result the *agent resultant magnitude* rises: This indicates 'something' is confining the swarm's distribution. This analysis could be used in identifying effective swarm distribution for the coverage of a sensor array as discussed by Ramaithitima et al. [121]

4. SWARM TYPE IDENTIFICATION

This chapter applies the metrics defined in chapter 3 to identify how the cohesion and repulsion field effects of the *interaction vector* affect the internal movement and the vector magnitudes between agents in a swarm.

There are two distinct inter-agent structures that can emerge in a boid-based swarm, *hexagonally-connected* or *hyper-connected*. These two swarm types are the result of the cohesion field effect detecting immediate neighbours only and when the neighbour field effect range extends beyond immediate neighbours to include additional agents.

If the goal is to maximise the coverage of an area by a swarm's agents then a hexagonal lattice is the most appropriate structure. In a *hexagonally-connected* swarm agents have visibility only of their immediate neighbours and are unaffected by agents beyond those neighbours. This effect can be implemented by ignoring agents beyond the initial neighbours detected or confining the field effects such that the connections do not occur. If the field effects extend beyond the immediate neighbours such that further agents are detected then there will be additional vectors affecting the calculations of an agent's *interaction vectors*. These additional vectors cause the structure to change logically from a lattice to a mesh. A mesh structure is a *hyper-connected* swarm.

4.1 Internal movement testing (static swarms)

To evaluate the metrics simulation parameters (field effects) need to be created such that they generate the two swarm types (Table 4.1).

Table 4.1 shows the parameter requirements for two swarm types. The parameters in the *Hexagonal* column generate a swarm structure where an agent can only detect immediate neighbours. The parameters in the *Hyper* column allow agents to detect agents beyond their immediate neighbours and therefore create additional neighbour connections which results in a hyper-connected swarm.

Weight component	Hexagonal Swarm	Hyper connected	Description
Sample Rate	100	100	ms - Unit sampling interval
k_c	5	5	weight adjuster for cohesion bias
k_r	15	15	weight adjuster for repulsion bias
k_d	0	0	weight adjuster for directional bias
Cohesion field	50	60	units
Repulsion field	40	40	units
Speed	20	20	units/s

Tab. 4.1: Swarm Weighted Model

The two sets of parameters are simulated using a swarm of 200 agents randomly distributed in an environment. The simulation generates data as described in § 2.13. The data extracts contain the distances and inter-agent magnitudes (*interaction vectors*) produced by the parameters.

4.2 Hexagonal swarm analysis

In a hexagonal swarm the field effects cause the agents to form a regular lattice. All the agents tend towards an even distributed with similar distances between each agent and its neighbours. In a well structured deployment, the agents in the swarm will show limited variation in the inter-agent distances and the *interaction vectors*. A perfect distribution is very unlikely in a swarm of agents due to the constant movement of the agents adjusting their positions to obtain an optimum position and the agents moving at a constant speed.

4.2.1 Distance based metric

The distance analysis graph (Figure 4.1) for the hexagonal swarm (using the parameters in Table 4.1, repulsion field 40 units, cohesion field 60 units) shows the distance metric being applied to the swarm over a period of 200 cycles. The graph shows a trace of the distance with the standard deviation displayed as error bars above and below the mean.

The swarm is initially (0-20 cycles) in a state of disorganisation, where the agent distribution is varied. The swarm then enters a phase where the hexagons are forming and the swarm starts to stablise (20-50 cycles). After about 50 cycles the field effects have stabilised the swarm structure and the swarm settles to a more stable state for the given set of parameters. The swarm then fluctuates as the residual internal movement maintains the swarm's structure. At this point the internal movement (jitter) is the 'background noise' generated by the field effects to maintain the swarm's structure.

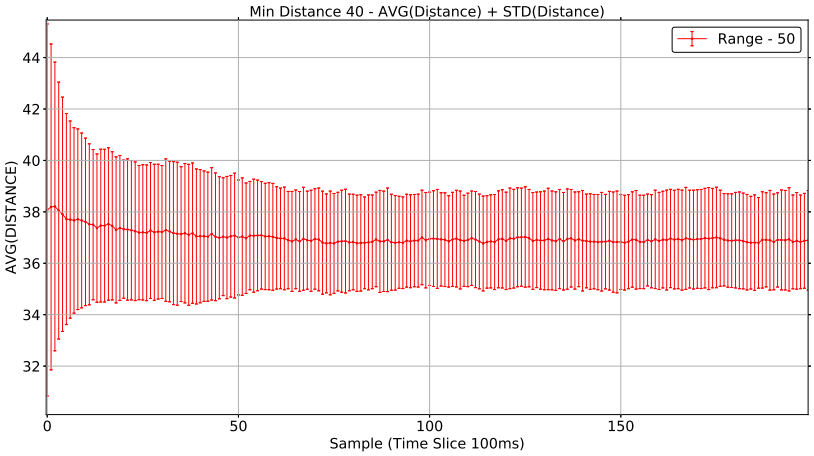


Fig. 4.1: Hexagonal swarm - distance metric

Figure 4.2 shows the distribution of the inter-agent distances for the duration of the simulation. The data forms a bell shaped distribution with a mean distance of approximately 37 units, the mean distance is the average of all the inter agent distances as shown in Figure 4.1. The graph shows the changes in the distribution of distances based on the aggregation of the whole simulation.

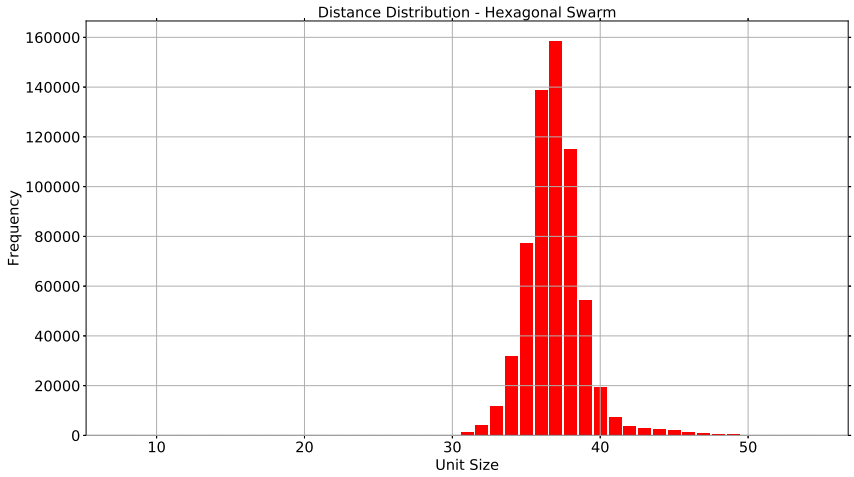


Fig. 4.2: Distance distribution

Figures 4.3 and 4.4 show the distributions at each time cycle and shows the changes in the distribution of the distances as the agents coalesce into a stable structure. Figure 4.3 shows the initial distribution for the time intervals from 0 to 100 cycles (10 seconds). As the cycles progress the mean increases and the standard deviation reduces as the inter-agent distances equalise. Figure 4.4 shows the final state of the swarm after 10 seconds.

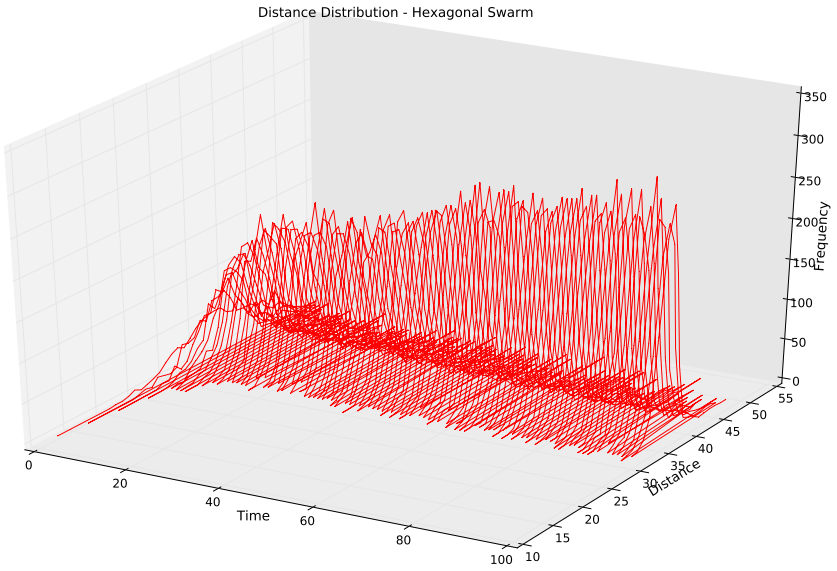


Fig. 4.3: Distance distribution / Time 0-10 seconds

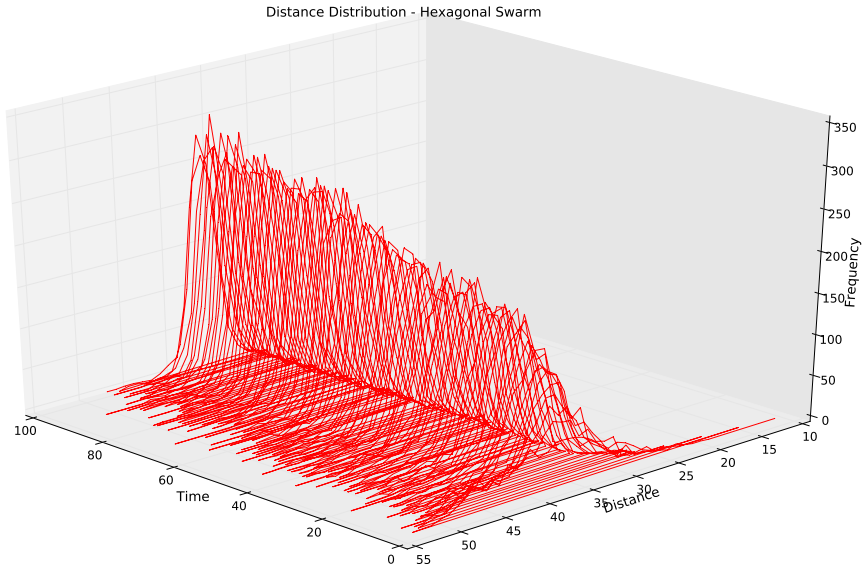


Fig. 4.4: Distance distribution / Time 10-0 seconds

4.2.2 Agent resultant magnitude (*interaction vector*) based metric

The distribution of the *interaction vector* magnitude can be plotted in the same manner as the distances. Figure 4.6 shows the distribution of the agents based on the *interaction vector* magnitude for the entire duration of the simulation. As with the distance based metric the data forms a normal bell shaped distribution with a mean magnitude evolving in time as shown in Figure 4.5. Figure 4.6 includes negative magnitudes, this indicates that sections of the swarm are expanding.

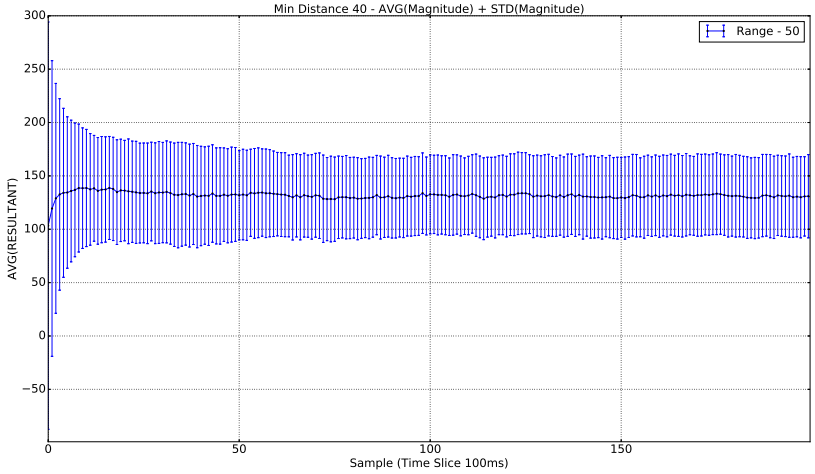


Fig. 4.5: Hexagonal swarm - Agent resultant magnitude metric

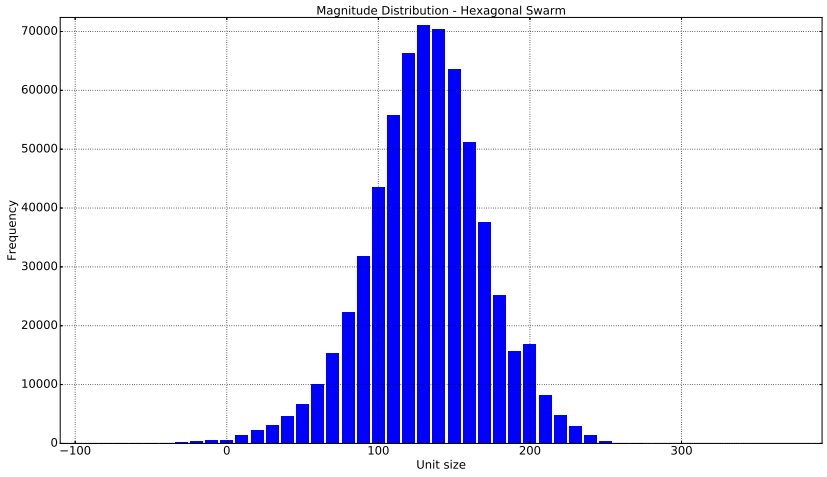


Fig. 4.6: Agent resultant magnitude distribution

Figure 4.7 and 4.4 show the distributions at each time cycle and show the change in the distributions of the magnitudes. As with the distance graphs Figure 4.7 shows the distribution at time interval 0 to 100 and Figure 4.8 shows the final state of the swarm after 10 seconds (100 cycles).

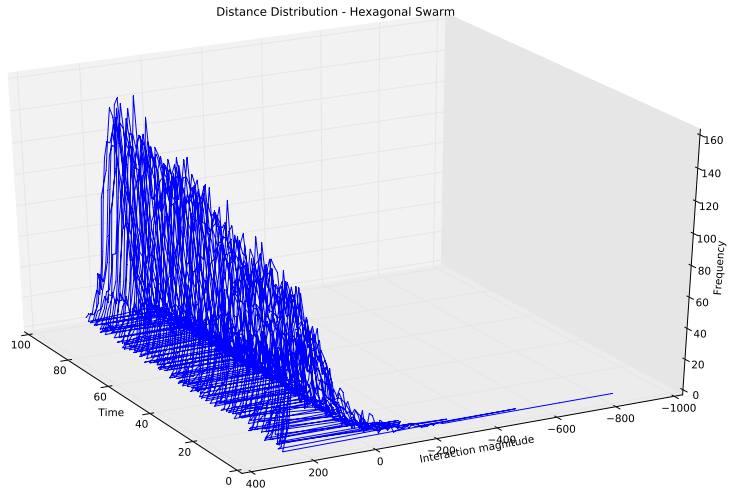


Fig. 4.7: Agent resultant magnitude distribution / Time 0-10 seconds

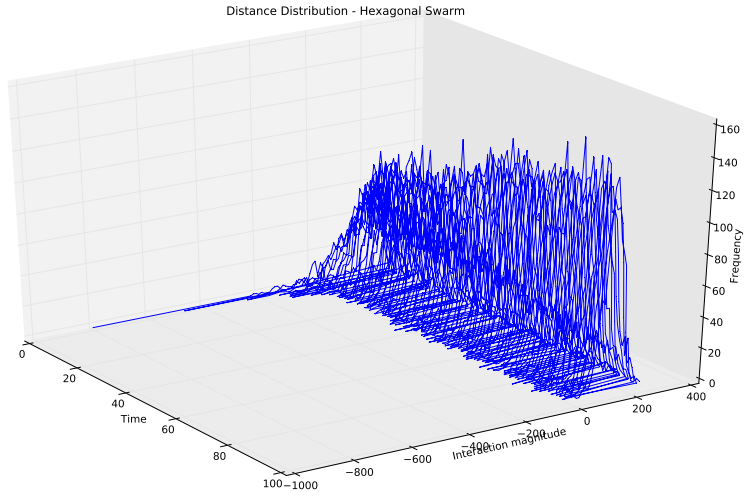


Fig. 4.8: Agent resultant magnitude distribution / Time 10-0 seconds

The hexagonal structure is the most stable structure [50] and can be classed as a swarm's most efficient state as the swarm is maximally distributed with agents having minimal or no cross connected agents. These results show that the field effects are producing a swarm that will tend towards having all distances equal which will produce the hexagonal effect as shown in Figure 2.8 on page 23.

4.3 Hyper-connected swarm analysis

When the field effects create a hyper-connected swarm the inter connectivity of the agents create a multi-modal distribution of the inter-agent distances. Figure 4.9 shows the inter-agent distances highlighted, near neighbours in green and extended neighbours in red. This is detectable in terms of how the internal movement metrics present these distributions. A hyper-connected swarm has a high level of cohesion causing the swarm to become very inflexible. The swarm appears 'stable' in terms how the overall structure is maintained (Figure 4.10), however, there is a greater variation in the *interaction vector* magnitudes, and resultant distances, than in a hexagonal swarm. The distances will maintain a good sound structure but the standard deviation from the mean is high. This elevated standard deviation (Figure 4.11 and 4.15) indicates that the swarm is not at its optimum distribution as the swarm's agents could be distributed further covering a

greater area without causing the swarm to break up. This can be achieved by increasing the repulsion field effect. In some circumstances this hyper-connected structure may be a desirable configuration to create a more 'rigid' platform: for instance to provide a close proximity wireless sensor network with multiple routing pathways. The connected distribution that causes the high standard deviation can be seen in Figure 4.12. There are two distinct peaks in the inter-agent distribution at approximately 38 and 58 units

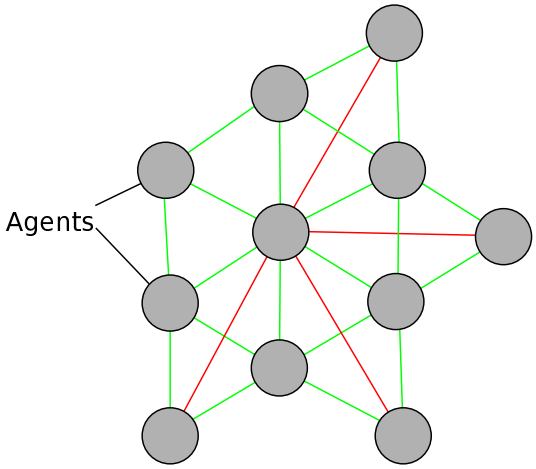


Fig. 4.9: Inter-agent links in a hyper-connected swarm

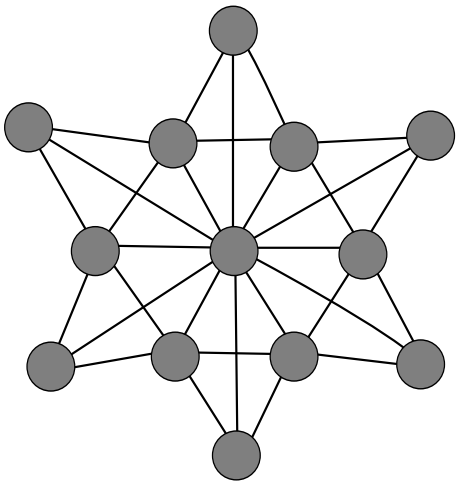


Fig. 4.10: Hyper-connected structure

4.3.1 Distance based metric

The distance based analysis graph (Figure 4.11) for the hyper-connected swarm shows the metrics being applied to the swarm over a period of 200 cycles.

The swarm is initially in a state of disorganisation and the average distance over the first 20 cycles shows the swarm compressing as the average distance falls. The swarm then enters a phase where the mesh structure forms and the swarm starts to stabilise. After about 100 cycles the field effects have resolved and the swarm structure settles to its most stable state for the given set of parameters. As with the hexagonal swarm the hyper-connected swarm's internal movement fluctuates to maintain the swarm's structure.

Fig. 4.11: Distance metric

Figure 4.12 shows the distribution of the agents over the entire simulation. By looking at the data in terms of the distribution graph it is possible to identify why the standard deviation is greater than that of the hexagonal swarm. The field effects, in this simulation, have created a bi-modal hyper-connected swarm. A bi-modal swarm is created when the agents in a swarm have the cohesion field effect set such that agents are neighbours one level further out from the immediate neighbours as shown in Figure 4.10. The result of this type of connectivity is that the distribution of the agents distances will produce two peaks in the distribution graph.

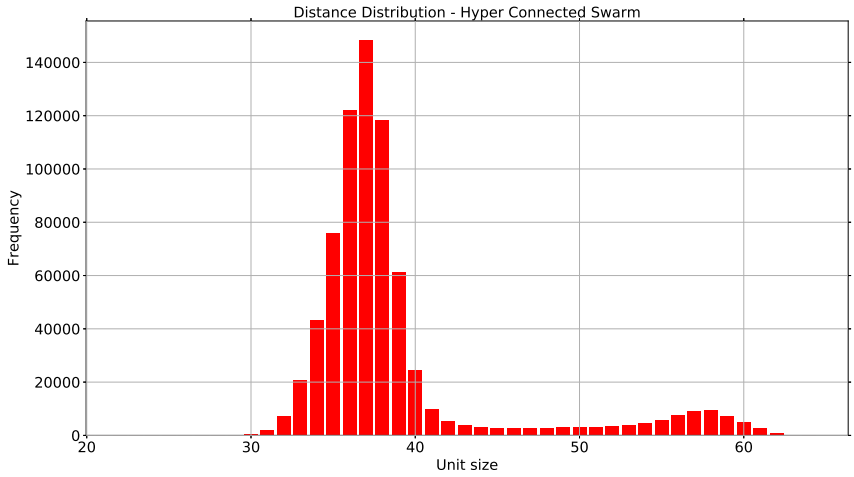


Fig. 4.12: Distance distribution

Figure 4.13 shows the initial distribution of the agents and the progression of the distribution of the swarm agents until 10s into the simulation. The initial state of the distribution at 0s is the same for both swarm types. The impact of the field effects are immediate as the swarm stabilises to a bi-modal distribution. Figure 4.14 shows the distribution of the data at 10 seconds (100 cycles) showing the resultant bi-modal frequencies the swarm is therefore a hyper-connected swarm.

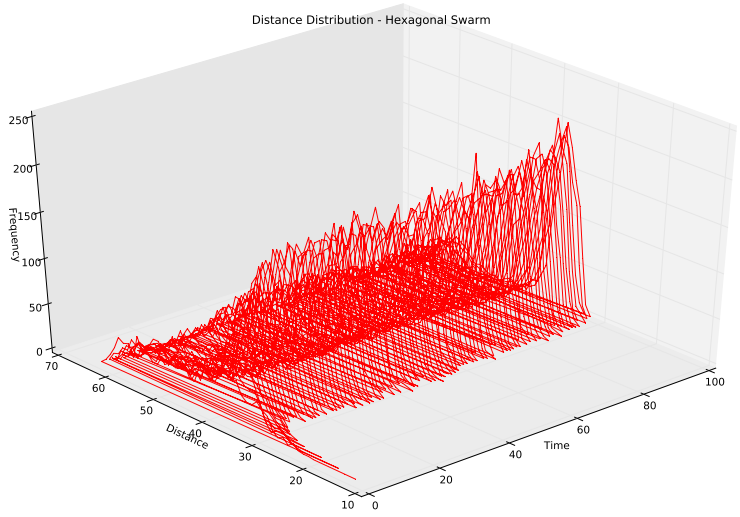


Fig. 4.13: Distance distribution / Time 0-10 seconds

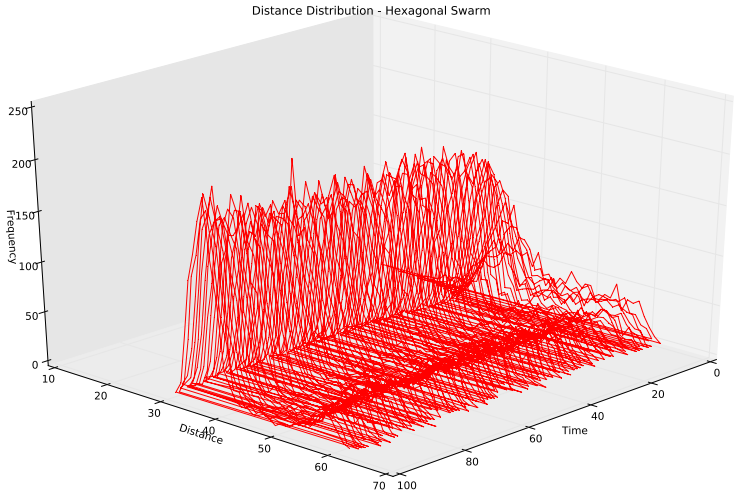


Fig. 4.14: Distance distribution / Time 10-0 seconds

Although the field effects in this simulation have produced a bi-modal distribution increasing the neighbour distance will create further swarm types that will be multi-modal as more distant agents are identified as neighbours.

4.3.2 Agent resultant magnitude based metric

The distribution of the potential magnitudes can be visualised in the same way as the distances. Figure 4.16 shows the distribution of the agents based on potential magnitude for the entire duration of the simulation. As with the distance based metric the data forms a bi-modal distribution with a mean magnitude as shown in Figure 4.15. Figure 4.16 shows negative magnitudes which indicates sections of the swarm are expanding.

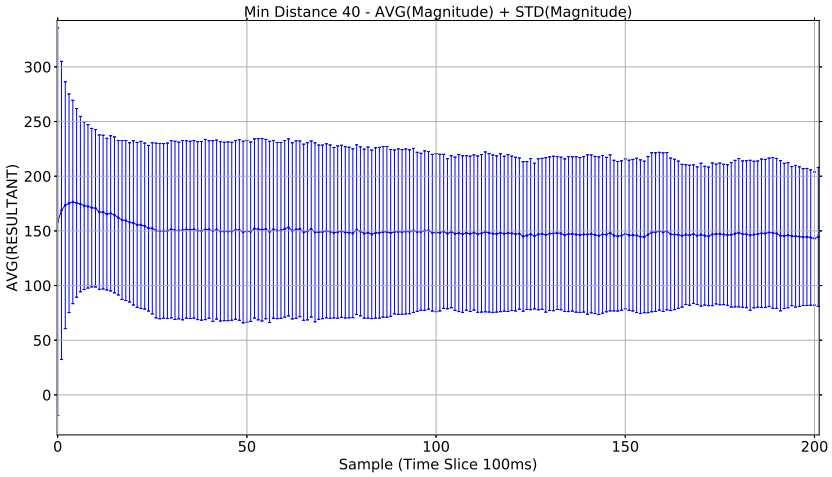


Fig. 4.15: Agent resultant magnitude metric

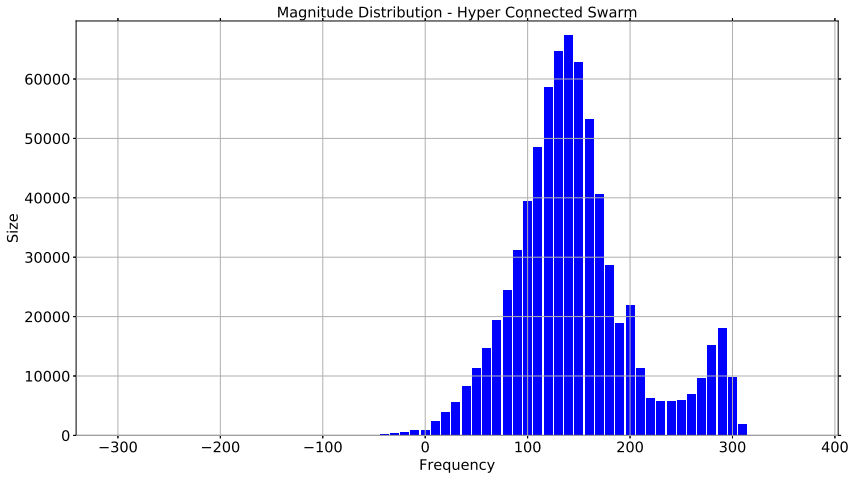


Fig. 4.16: Agent resultant magnitude distribution

Figure 4.17 and 4.18 show the distributions at each time cycle and visualise the change in the distributions of the agent resultant magnitudes. As with the distance visualisations Figure 4.17 shows the initial distribution at time interval 0 to 100 and Figure 4.18 shows the final state of the swarm after 10 seconds (100 cycles). Both show the bi-model state of the swarm emerging.

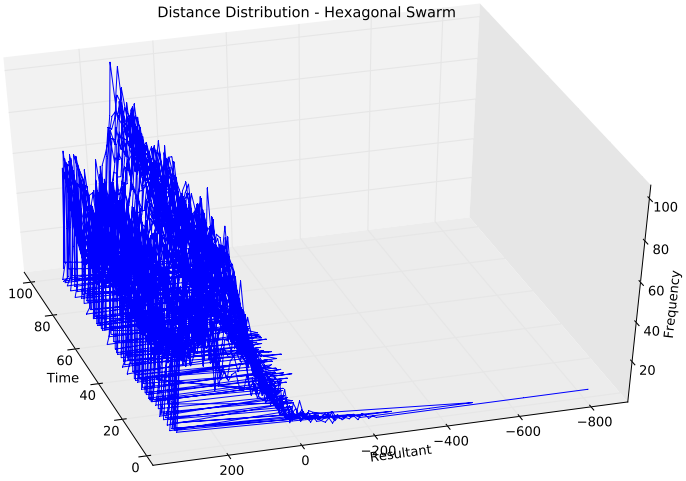


Fig. 4.17: Agent resultant magnitude distribution / Time 0-10s

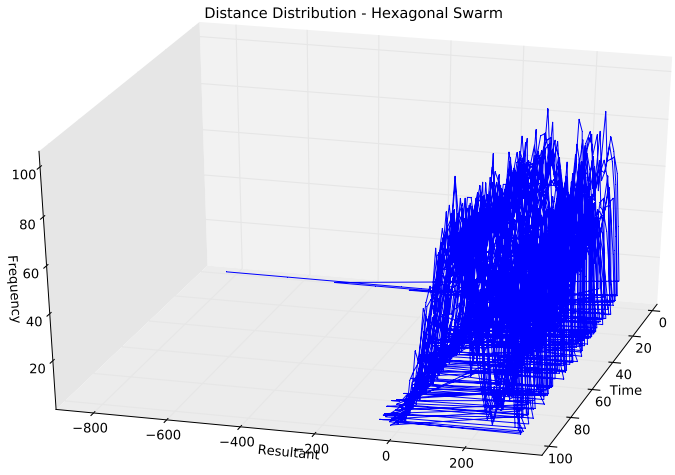


Fig. 4.18: Agent resultant magnitude distribution / Time 10-0s

4.4 Metric comparison

Looking at the two swarm types together the characteristics of the swarm types present themselves as change in the standard deviation from either the average distance or the average magnitude of the agents in the swarm.

Figure 4.19 shows the swarm with the two different field effects for cohesion. The metric used in this analysis is the distance between agents. The result shows that the deviation on the hyper-connected swarm (shown in green) has a higher standard deviation than the hexagonal swarm (shown in brown). This is caused by the bi-modal nature of the hyper-connected swarm.

Figure 4.20 shows the analysis of the agent resultant magnitude between the agents in the swarm which demonstrates the same characteristic emerging.

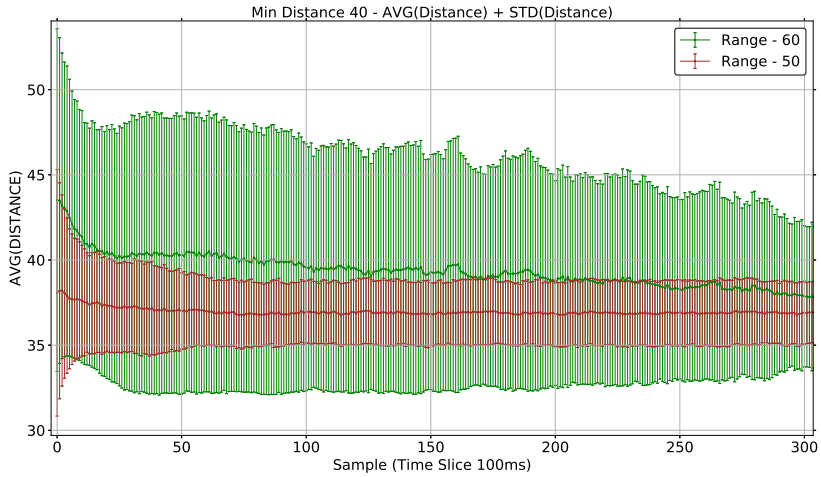


Fig. 4.19: Distances metric comparison

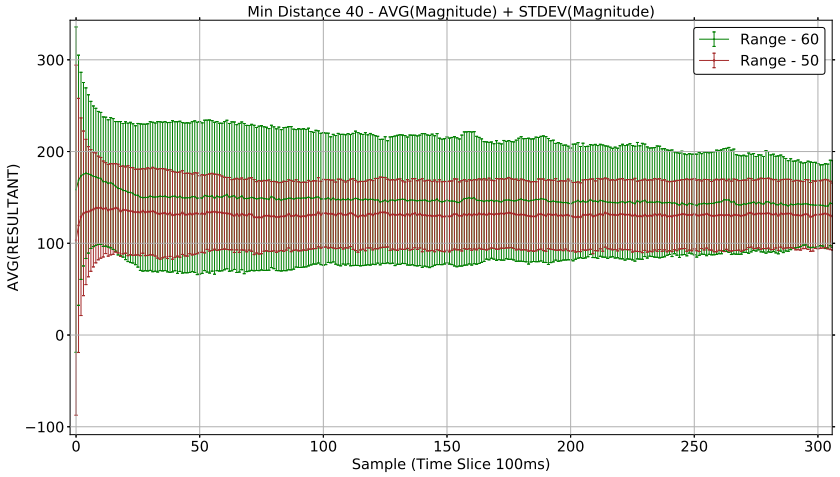


Fig. 4.20: Agent resultant magnitude metric comparison

A well structured and balanced swarm should therefore have a very low standard deviation in terms of the resultant metric. Figures 4.21 and 4.22 clearly show using either magnitude or distance as the metric that the deviation from the mean can highlight the two different types of swarm when using the same repulsion field setting.

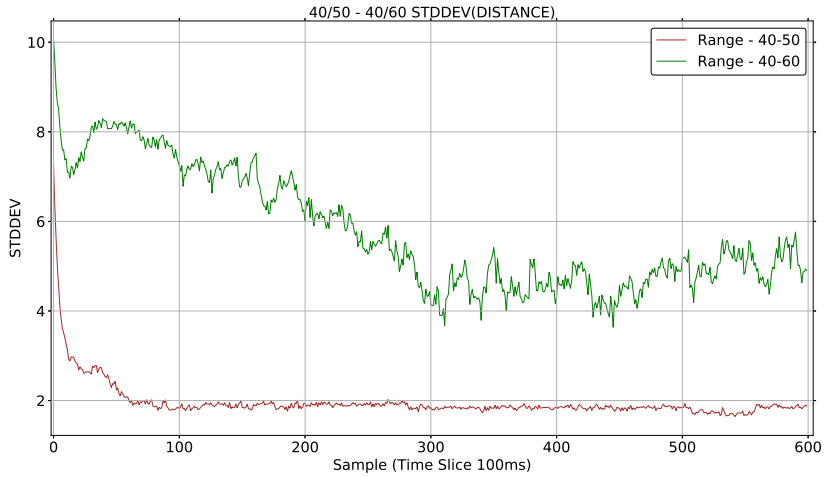


Fig. 4.21: Distance based metric

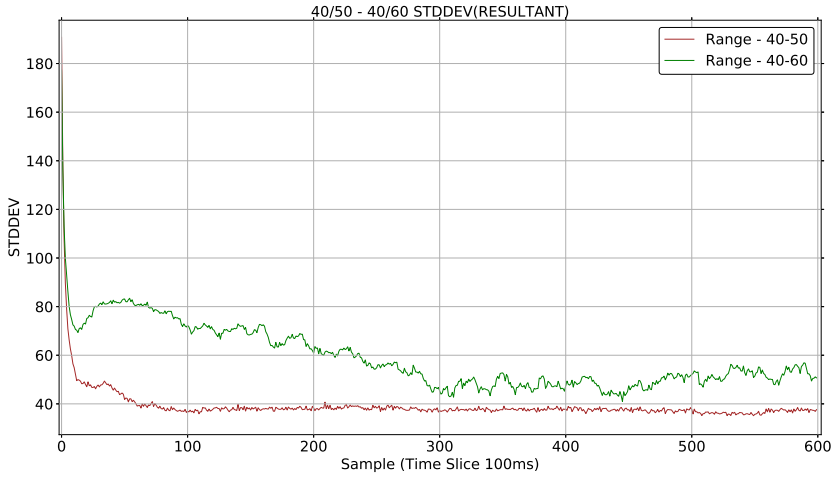


Fig. 4.22: Agent resultant magnitude based metric

The distribution of the agents over the entire simulation also shows it is possible to identify the mode of the swarm that is being generated by the cohesion field effect of the swarm.

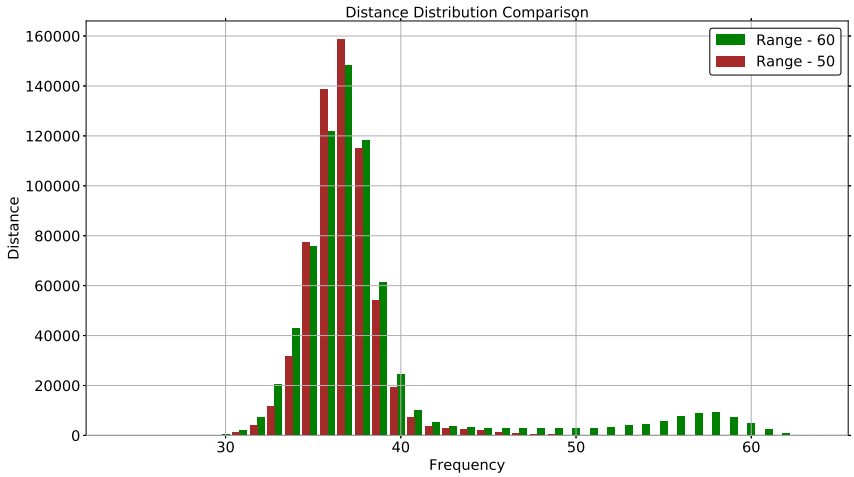


Fig. 4.23: Distance comparison

Figure 4.24 and 4.25 show that over the duration of the simulation that the frequency of the mean for a hexagonal swarm is greater than that of the bi-model swarm. This is expected due to the additional vectors that are generated between agents.

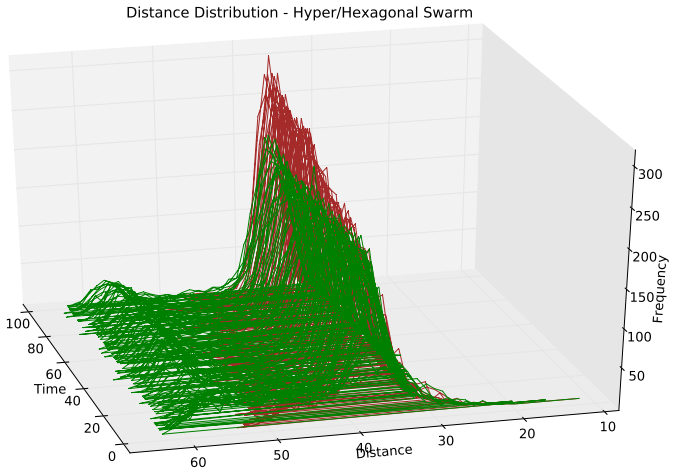


Fig. 4.24: Distance comparison / Time 0-10 seconds

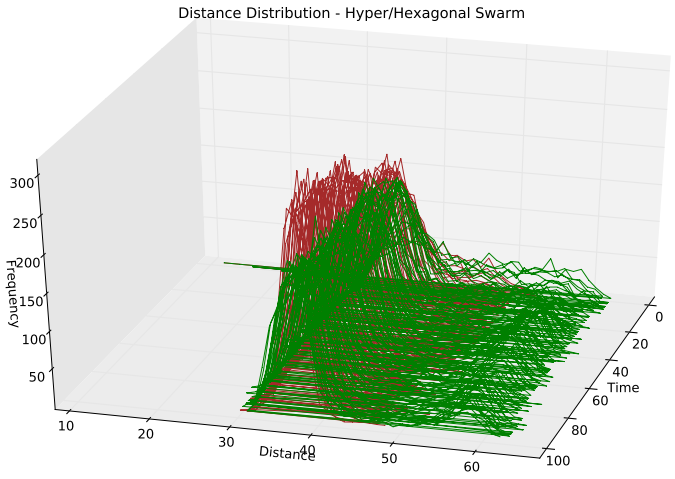


Fig. 4.25: Distance comparison / Time 10-0 seconds

Figure 4.26 shows that the mean magnitude potential for a hexagonal swarm is lower than that of the hyper-connected swarm.

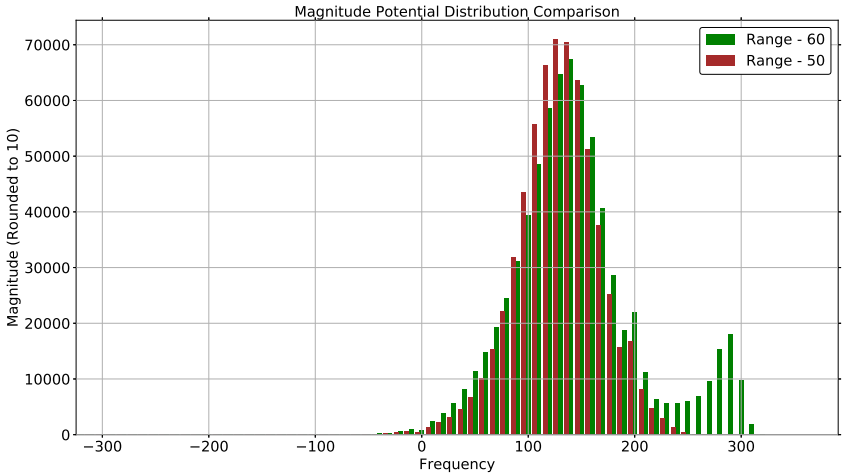


Fig. 4.26: Agent resultant magnitude comparison

The most notable difference between the two magnitude metric analyses (Figure 4.27 and 4.28) is the initial potential magnitude of the hyper-connected swarm being greater

than the hexagonal-swarm. This is due to the additional range of the field effect which generates a larger initial magnitude potential drawing the swarm together.

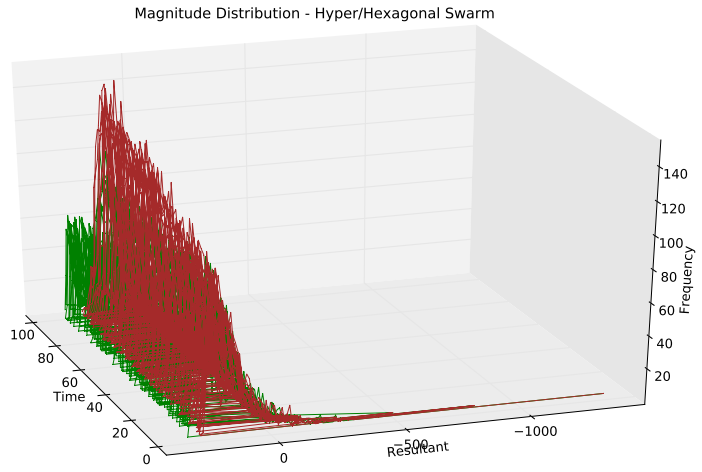


Fig. 4.27: Agent resultant magnitude comparison / Time 0-10 seconds

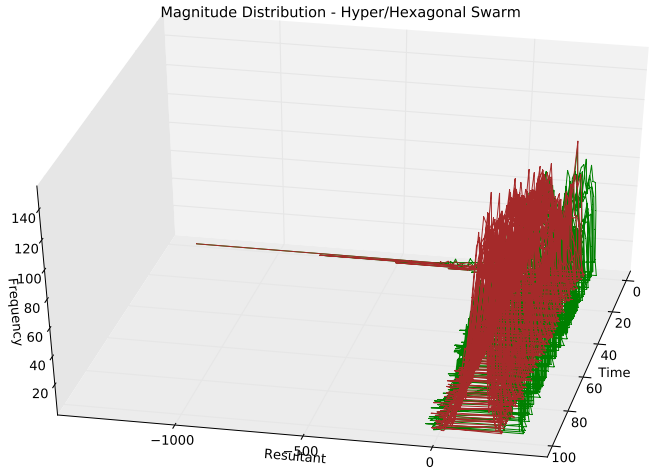


Fig. 4.28: Agent resultant magnitude comparison / Time 10-0s

4.5 Static swarm conclusion

These results in terms of using magnitude or distance, provide similar profiles in terms of the changes in the profile of the plotted results. The distance metric discussed by Navarro [104] is used by many researchers that analyse agent distribution such as Cheein et al. [21], Bennet et al. [14] and Barnes et al. [11, 10, 12, 13]. Gazi et al. [39, 40, 38, 41] use this metric to analyse swarm stability.

The similar profiles of the two metrics shows that either mechanism is suitable for identifying the internal movement within the swarm and providing a mechanism to compare the effects of swarming algorithms. The magnitude analysis, however, also provides a mechanism to determine if the swarm is expanding (a negative magnitude) or cohesive (a positive magnitude).

The metrics both provide an ability to adjust the variable parameters of the swarming mechanics (cohesion, repulsion and direction) or the range of the neighbour detection and minimum proximity distances and to identify the effect that those changes have upon the swarm in terms of the swarm's internal movement. If we consider the internal movement to be a measure of the quality of a swarm in a given situation then these measures are identifying the swarming algorithms effectiveness.

4.5.1 Arbitrary sized swarms

This thesis if focused on arbitrary sized swarm's. The algorithms that are used only require localised sensing which therefore all movement and positioning calculations are based on neighbours. As localised data is used it should be possible for the algorithms to be effective with swarms of any size. The swarm's configuration in this section is based upon a swarm of 200 agents but the application of the field effects coordination will work with larger swarms. Figure 4.29 shows a comparison the distance metric of swarms of 200 and 500 agents and Figure 4.30 shows a comparison of the Magnitudes.

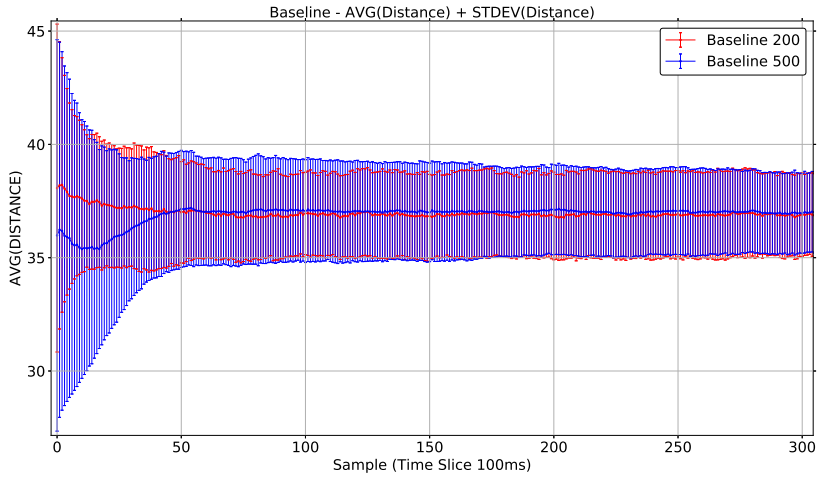


Fig. 4.29: Swarm size distance comparison

Figure 4.29 shows the average distance of the agents fluctuate while the swarm stabilises (disorganised stage) and eventually both swarms stabilise to comparable distances and variations.

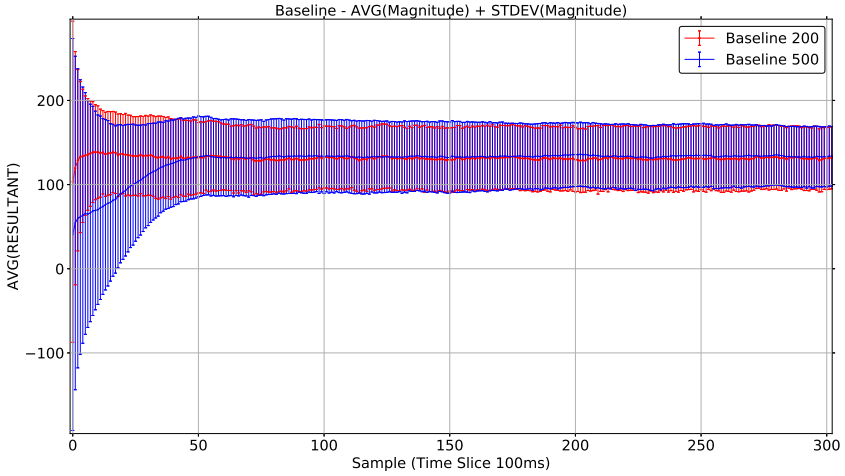


Fig. 4.30: Swarm size magnitude comparison

Figure 4.30 shows the average magnitude of the agents fluctuate while the swarm stabilises (disorganised stage) and eventually both swarms stabilise to comparable magni-

tudes and variations.

These two metrics show that the swarm is able to expand to its maximum distribution without being impeded. If the swarm's area was impeded the magnitude would not have been able to stabilise and would have remained high. The identification of swarm bound containment will be discussed in chapter 7.

5. SWARM COORDINATION - PERIMETER DETECTION

This chapter identifies the effect a *destination vector* has upon the motion of a swarm. Three coordination techniques are applied to a swarm and the metrics defined in chapter 3 are used to identify the effects. The three coordination techniques are: full perimeter detection, where only edge based agents will have a *destination vector* applied, partial perimeter detection, where a subset of the perimeter agents are detected and finally all agents in the swarm having a *destination vector* applied. For all the experiments the base parameters of the swarm are fixed ensuring that the only variable will be the *destination vector* that is applied by the coordinating agents. This process allows the impact of the algorithms to be isolated and compared. Section 5.8.5 discusses the potential variations to the base parameters and the effects this has on controlling the impact of the weightings.

5.1 Baseline specification

To allow an analysis of each algorithm's effect on a swarm a baseline measurement of 'background' variance (jitter) is identified. The baseline measurement provides a comparative data set for the experiments. The baseline data set is for a static swarm (no destination vector) with the same internal parameters.

Assumption 1: The swarm used for the experiments consists of 200 agents randomly distributed (Figure 5.1).
Assumption 2: The field effects and bias will be set as shown in Table 5.1 with the weight adjuster $k_d = 0$ for the baseline.
Assumption 3: All destination based experiments will have the weight adjuster set to $k_d = 100$.

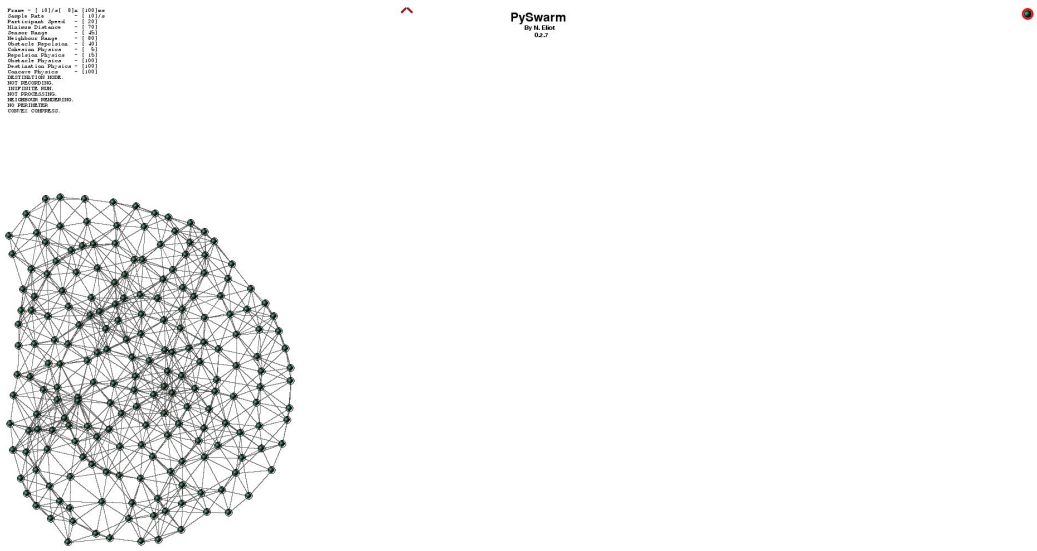


Fig. 5.1: Sample swarm 200 agents initial state (*screen shot from simulator*)

Weight Component	**Hexagonal**	**Description**
Sample Rate	100	ms - Unit sampling interval
k_c	5	weight adjuster for cohesion vector
k_r	15	weight adjuster for repulsion vector
k_d	0/100	weight adjuster for destination vector 0 for static baseline 100 for destination based
Repulsion field	70	units
Cohesion field	80	units
Speed	20	units/s

Tab. 5.1: Swarm Weighted Model

The results for the baseline experimental swarm are shown in Table 5.2 and 5.3. The metrics used are the distance [104] and resultant inter-agent magnitudes (*inter-agent vector*).

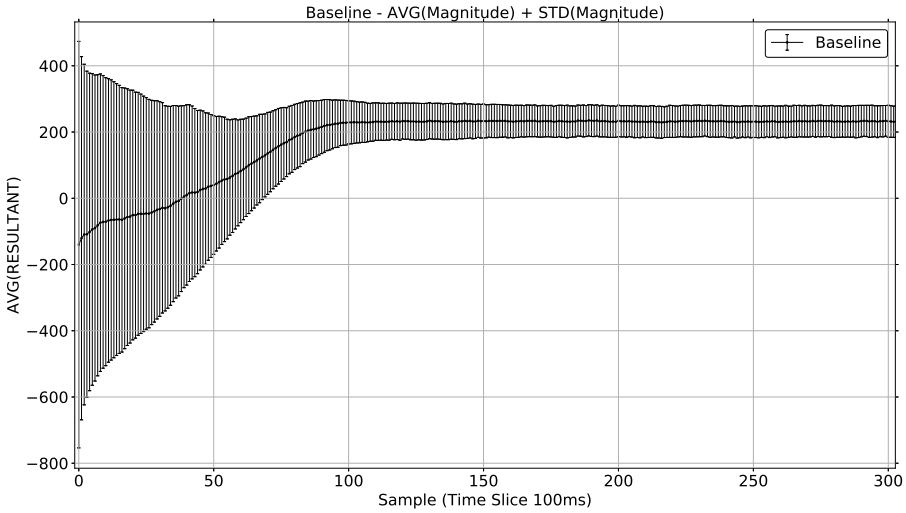


Fig. 5.2: Baseline internal movement - magnitude

Figure 5.2 shows the magnitude between t_0 and approximately t_{48} dominated by repulsion ($|k_r v_r| > |k_c v_c|$). This indicates that the swarm is undergoing a rapid expansion as the bias within the swarm shows a high level of repulsion. Between approximately t_{48} and t_{100} the magnitude is positive but rising which indicates the swarm is cohesive and will remain a cohesive unit but it is still expanding. At approximately t_{110} the swarm has completed it's expansion and has reached a settled state with the 'background' movement maintaining the swarm's structure with $|k_r v_r| < |k_c v_c|$. The residual 'jitter' is at its minimum for the configuration and the agent distribution is at its optimum for the cohesion and repulsion settings. The resultant $|k_r v_r| < |k_c v_c|$ is due to the cohesion field being larger than the repulsion field which prevents the swarm 'breaking up'.

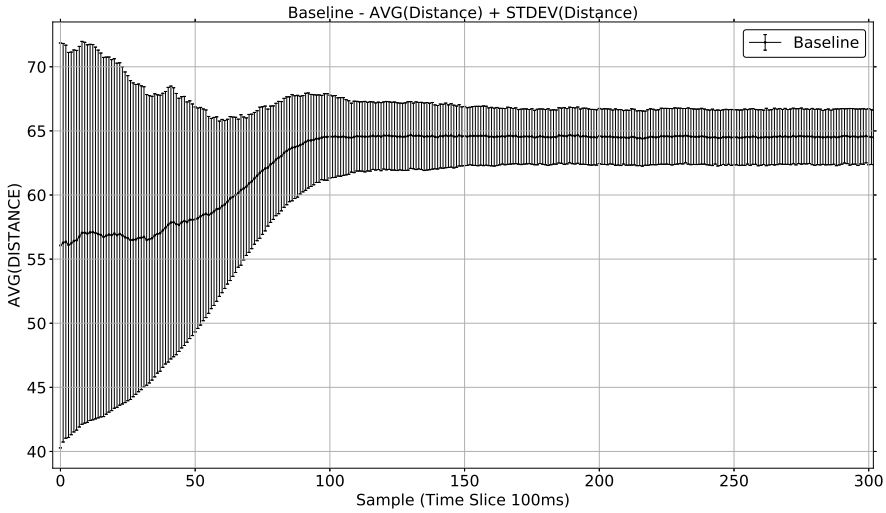


Fig. 5.3: Baseline internal movement - distance

Figure 5.3 shows the internal movement and the deviation from the mean for the swarm based on the inter-agent distances. There is no *destination vector* applied so all internal movement is being generated by the cohesion and repulsion vectors. The swarm is initially in a disorganised state but once the agents have expanded a stable hexagonal formation evolves. Figure 5.3 shows the first 30s of the simulation which is sufficient time for the swarm to 'settle' and transform to its most stable state. Due to the metric only showing inter-agent distances it is not possible to determine if the swarm will remain cohesive.

The centroid path for the baseline swarm is shown in Figure 5.4. Due to there being no *destination vector* for the swarm the centroid changes positions based on the inter-agent positional changes caused by the *interaction vectors*. These changes are the result of each agents attempting to attain a state of equilibrium within their 'clusters'.

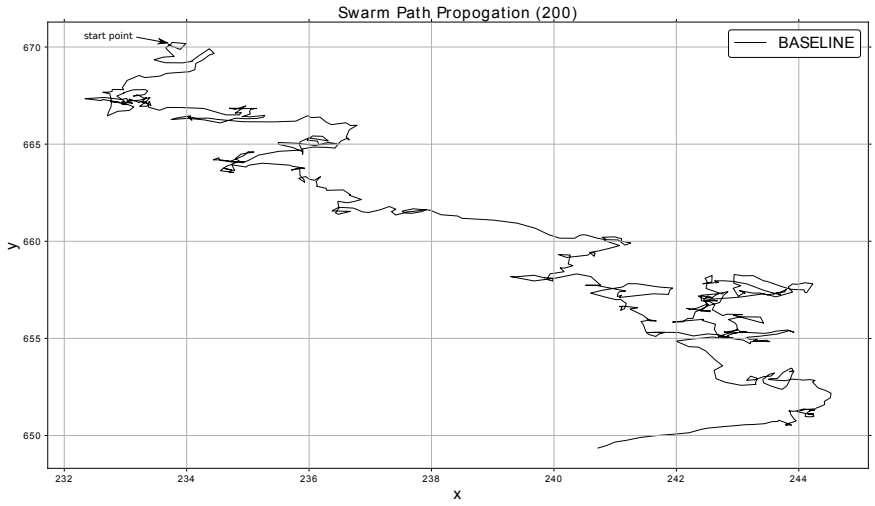


Fig. 5.4: Baseline swarm path

These baseline measurements are used to identify the changes that occur in the structure of a swarm when changes are made to the weightings of the field effects, the number of coordination agents and the *destination vector* weighting.

5.2 Destination vector application

Many mechanisms can be used to influence a swarm to travel in a specific direction. These mechanisms generally involve the use of sensors such as magnetometers or GPS's to achieve a directional coordinate [142, 150].

This thesis aims to produce algorithms that will reduce the number of active sensors that are needed for the directional coordination of a swarm. When reducing the number of active sensors the algorithms must maintain sufficient coordination of agents to influence the swarm's direction and therefore create an overall *destination vector* to a swarm's movement. Reducing the number of active sensors will reduce the net energy usage of the sensors and potentially increase the sensors viable lifetime. Another aspect of the algorithms is that they should minimise the internal disturbance (jitter) created by the *destination vector*.

There are two distinct approaches to configuring agents: all the agents are configured

identically (homogeneous swarm) or the agents are in a selection of different configurations (heterogeneous swarm) [9].

The following assumptions are made in this thesis:

Assumption 1: All agents are identical (homogeneous) but are able select subsystems as necessary and adopt different roles depending upon the algorithm being used to coordinate the swarm.
Assumption 2: As the swarm progresses towards a destination the role of an agent can change such that all the agents are identical in construct but they can enable and disable sensors as necessary. This will allow agents within the swarm to self promote to the coordinator role should a set of conditions arise.
Assumption 3: All agent in the swarm are autonomous and each agent has its own power supply.

Reducing the usage of sensors will reduce the net power consumption of the swarm and increase the time that an agent can be a part of a swarm. If the purpose of the swarm is reconnaissance then the reduction in energy usage may allow a longer traversal time, or allow the swarm to travel further. As agents in the swarm will use their resources at different rates some agents may be lost due to resource exhaustion. The algorithms should therefore be resilient to this.

5.3 Swarm destination vector

The following assumptions are made in this thesis:

Assumption 1: The direction of the swarm is based upon having a fixed goal that is known to all the agents (§ 2.7).
Assumption 2: The goal is migrated towards based on a simulated GPS signal and the bias k_d is applied to the *destination vector* to influence the movement of the agents.

The swarm's destination is established by the coordinator agents influencing the non-coordinator agents. This influence is applied through the coordinators proximity to non-coordinator agents (cohesion). Coordinator agents are identified by a set of conditions. The conditions are the activators for the coordinator identification algorithms.

5.4 Identifying the coordinator role

For an agent to change its role from a standard agent to a coordinator there needs to an algorithm that identifies a set of conditions to trigger the change. This same algorithm also triggers an agent to revert to being a standard agent. The coordinator selection algorithm is separate from the agent coordination algorithms discussed in chapter 2.

In this thesis three algorithms are presented and compared. In the first algorithm, all agents have the role of a coordinator permanently. This will be referred to as the all-agent algorithm. The other two mechanisms employ a selection criteria to identify if the simulated GPS sensor should be enabled. These two algorithms are: full-perimeter detection and the partial perimeter detection. Partial perimeter detection will be referred to as the basic-count algorithm.

There are many other ways in which the coordinator role could be established such as randomly enabling a GPS for a set period of time or oscillating the role on and off but they are beyond the scope of this thesis.

5.5 Monolithic swarm - (all-agent)

In a monolithic swarm [9] the propagation towards the goal is achieved by all the agents using a GPS signal to give each agents' movement a *direction vector* towards a required goal (Equation 2.7). There is no selection criteria and no computational overhead for the agents.

Figure 5.5 shows a screen shot from the swarm simulator with all agents GPS's enabled. This can be seen as the agents highlighted with a ring around them. The small 'tail' on each agent is the *movement vector*.

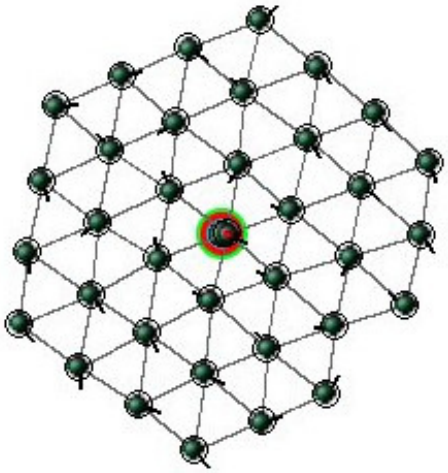


Fig. 5.5: Monolithic agents (Circles indicate all agents are using GPS, *screen shot from simulator*)

5.5.1 Baseline and effect of no perimeter detection

To identify the effect of the all-agents algorithm the simulation is executed using the same parameters as the baseline swarm with the added *destination vector* on all the agents. The effect of introducing this bias on the swarm's internal characteristics can be seen in Figures 5.6 and 5.7.

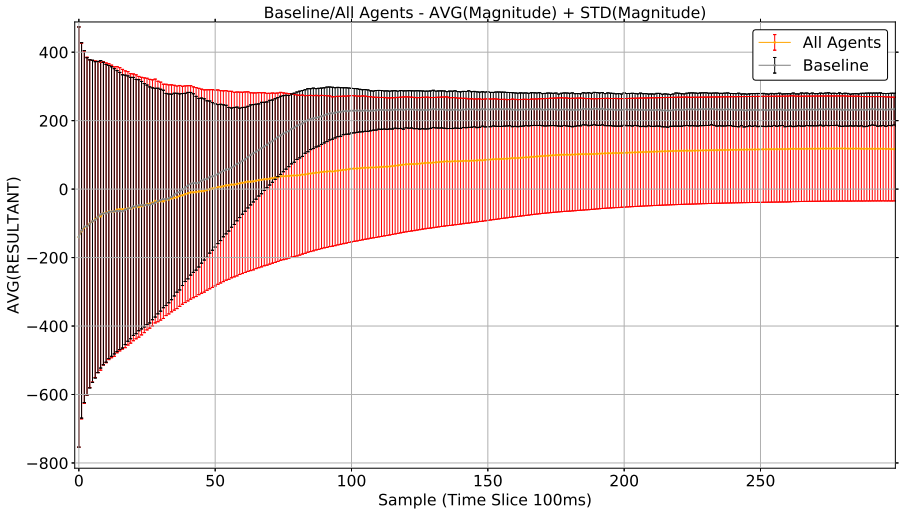


Fig. 5.6: Baseline/All agents comparison *inter-agent vector* magnitude

Figure 5.6 shows the effect on the *inter-agent vector* magnitude of the swarm: the average resultant magnitude is lower than the baseline. This is due to there being a greater distribution of *inter-agent vector* magnitudes from the disturbance caused by the *destination vector* on all of the agents and the swarm is therefore less cohesive. The graph also shows that there is a higher variation from the mean; again this is due to the change in the bias of all the agents. The agents are moving towards a goal and are therefore not moving to an equilibrium distribution. Figure 5.6 also shows that the *direction vector* increases the time it takes for the swarm to distribute the agents and although the deviation diminishes the swarm cannot overcome the *direction vector's* effect and the swarm appears disorganised.

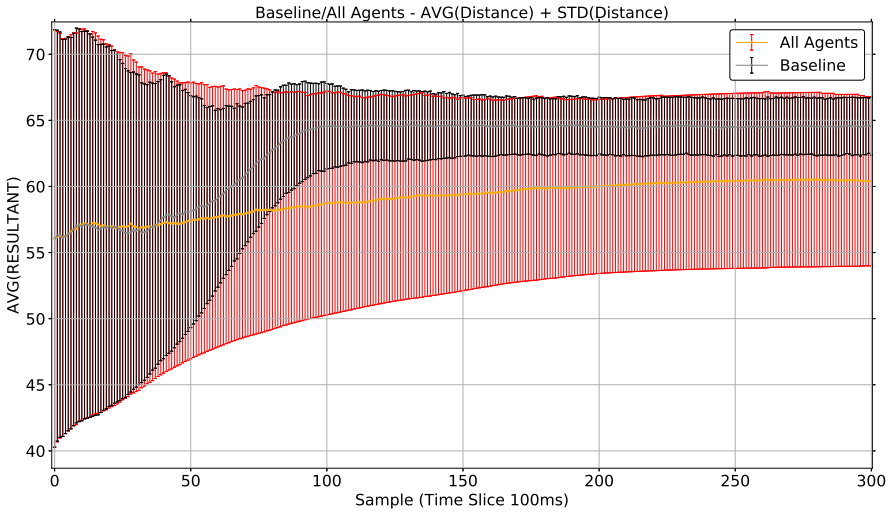


Fig. 5.7: Baseline/All agents comparison (distance)

Figure 5.7 shows the effect on the inter-agent distances. The average distance is lower than the baseline. This effect is caused by the *direction vector* reducing the effect of the repulsion field. The agents therefore compress slightly due to the cohesion.

A curious effect of having all the GPS sensors enabled is that as the swarm approaches the destination there is an increased compression effect and the internal disturbance increases. This is the result of the agents 'pulling' in a cone effect. The agents paths converge on the destination (Figure 5.8). This effect is caused by the number of agents being effected by a *direction vector*. This effect is less identifiable if a swarm's target is at a greater distance.

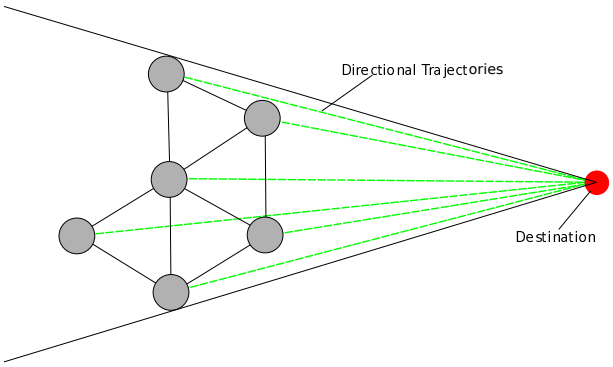


Fig. 5.8: Conical destination trajectories

This effect can be seen in both the magnitudes shown in Figure 5.9 and in the distances shown in Figure 5.10 which both show the averages initially decreasing but then increasing again as the swarm approaches the destination. This is accompanied by a greater standard deviation caused by the *direction vector* affecting the priority of the swarm as agents are drawn to the destination reducing the effect of the *inter-agent vectors* creating stable structures.

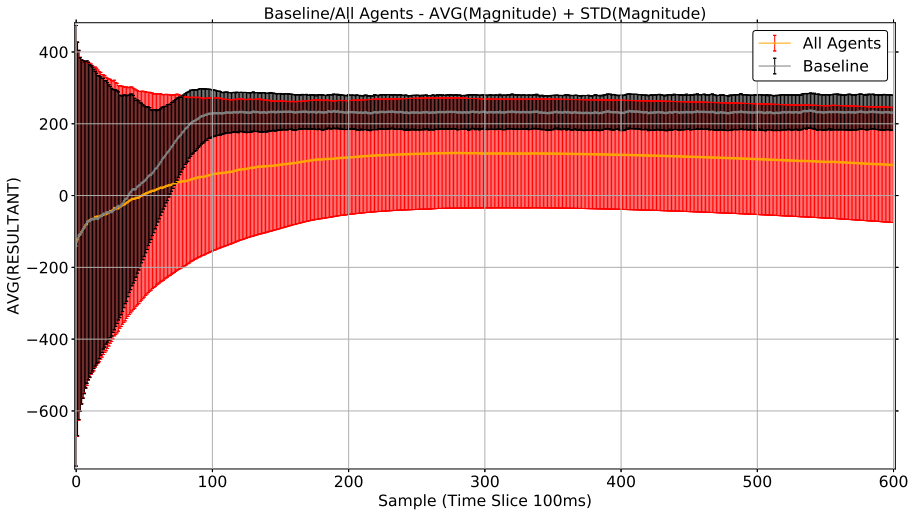


Fig. 5.9: Baseline/All agents *inter-agent vector* magnitude comparison (60 seconds)

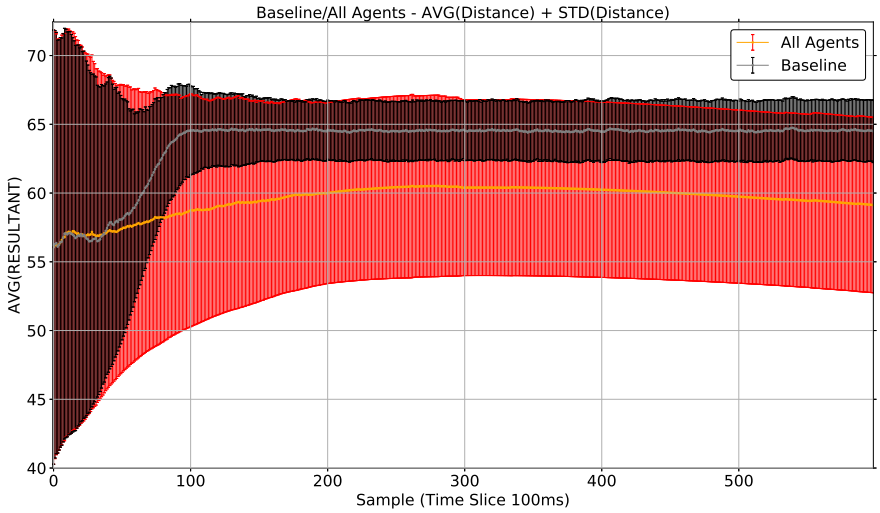


Fig. 5.10: Baseline/All agents distance comparison (60 seconds)

5.6 Simple multifaceted swarm (basic-count)

In a simple multifaceted swarm the propagation towards the goal is achieved by generating a *destination vector* for an agent when the agent conforms to a simple counting rule. This *destination vector* is obtained by using the agent's GPS to identify its current coordinate and generating a vector from that coordinate to a specific destination point. This rule creates a subset of agents that apply a local *direction vector* to their movement. This has the effect of creating a directional effect on the whole swarm. The aggregate *direction vector* magnitude of the swarm is less than that produced when all agents apply a *direction vector*. This reduction is further diluted by agents without the *direction vector* calculating a *movement vector* to an equilibrium position (Equation 2.7). This change in the aggregate *direction vector* will impact on a swarm's 'jitter'. This is discussed in more detail in § 5.8.5. With there only being a limited number of peripheral agents invoking their GPS signal there will also be a net reduction in the energy usage of the swarm (§ 5.9).

A swarm with the appropriate field effects will propagate towards hexagons made from a central agent and six neighbours (Figure 2.8). A swarm can be influenced by its surroundings, which involves the interaction of other agents (as neighbours) and obstacles

which can cause a compression of the agents. Compression can also occur during the stabilisation phase of a swarm. These situations can cause an agent to have less than 6 neighbours but still be surrounded. A general rule therefore to detect a rudimentary boundary of a hexagonal swarm is to calculate the number of neighbours an agent has. If the total is less than five ($nbr(b) < 5$) then there is a high probability that the agent is either in a void within the swarm or on an external perimeter. If an agent is on a perimeter or a void the it should enable its GPS and become a coordinator and provide a directional bias to the swarm via its *direction vector*. This coordinator role affects the immediate neighbours *movement vector* through cohesion/repulsion (Figure 5.13). This simple detection algorithm allows a directional bias to be induced into a swarm with minimal computational impact on the coordinator agents. The issue of voids and perimeters is discussed in section 5.7 and chapter 6.

5.6.1 Simple multifaceted algorithm

The mechanism to enable a coordinator agent is to count the number of neighbours that an agent has (Algorithm 1). This process is already part of the cohesion and repulsion calculations.

Algorithm 1: nbr(b)

Data: b, S

`/*` b `is an agent and` S `is a set of agents */`

`/* The set of agent could be provided by an omni directional camera or a set of ultrasonic sensors */`

$N \longleftarrow \emptyset$

foreach b' *in* S **do** `/*` b' `is each agent in` S `*/`

  if $length(bb') < D_b$ **then** `/*` D_b `is the neighbour range */`

    $N \longleftarrow N \cup \{b\}$

  end

end

`/*` N `is a set of Bots` $b_1, b_2, \ldots, b_n$ `*/`

return N

5.6.2 Basic-count effect

Figures 5.11 and 5.12 show the comparison of the baseline swarm against the goal-based swarm with reduced GPS usage. Both graphs show that the settling period of the swarm from the disorganised state is very much in line with the settling period of the baseline swarm. This is due to the algorithm's effect when the swarm is initially compressed. The neighbour count prevents any agents from being coordinators ($|nbr(b)| > 5$). This is supported by the propagation of the coordinator role in figure 5.32. The swarm therefore acts in a similar manner to the baseline. As the simulation progresses the swarm settles into a condition that is similar to baseline with a higher deviation and lower mean resultant *inter-agent vector*. This is due to there being a *direction vector* which impacts on the coordinator agents and propagates to the non-coordinator agents through proximity. This effect is shown in figure 3.3.

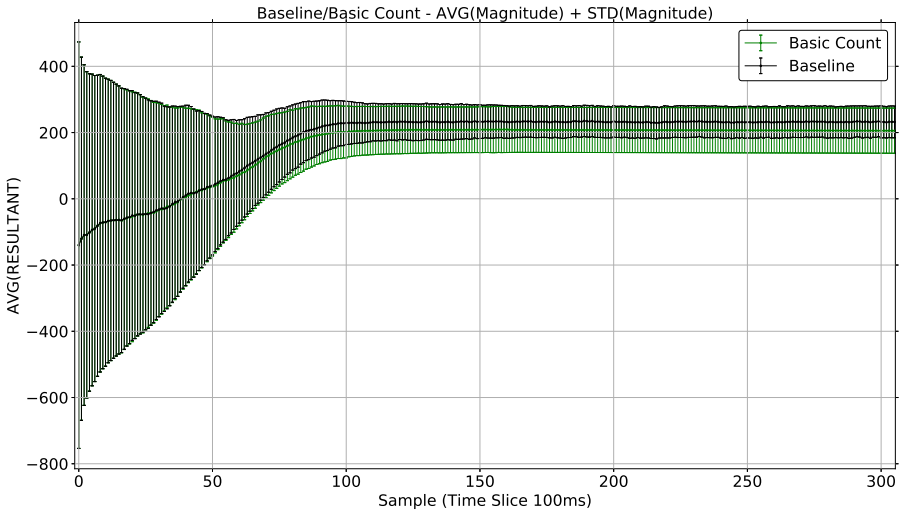


Fig. 5.11: Baseline/basic-count magnitude potential comparison

Figure 5.11 shows that initially the average *inter-agent vector magnitude* can be seen to follow the same trend as the baseline. This is expected as the algorithm's selection criteria is such that no GPS sensors are enabled when the swarm is compressed and the agents will have a neighbour count well above the trigger level for the coordinator role. When the swarm expands sufficiently the trigger level is met by some agents and there is a gradual increase in the number of coordinators. This also creates a greater distribution

of resultant *interaction vectors*, shown as the change in the standard deviation.

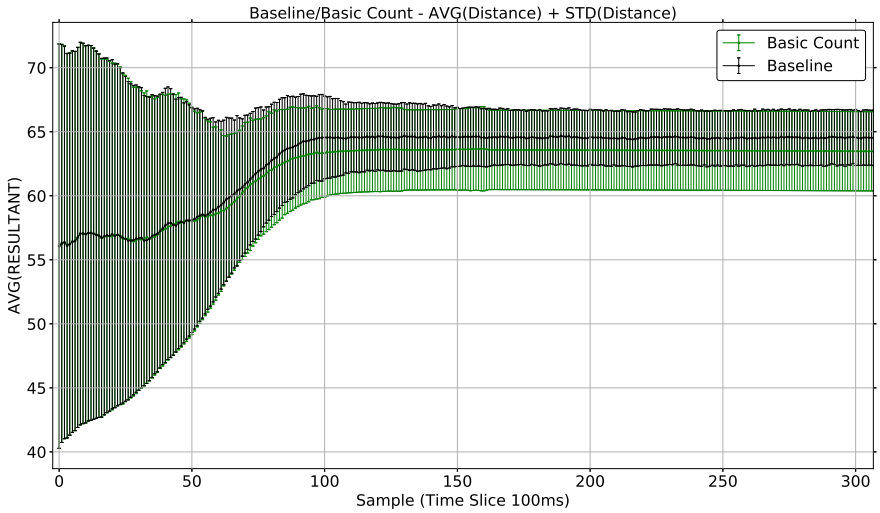


Fig. 5.12: Baseline and basic-count distance comparison

Figure 5.12 shows that the impact of the directional bias allows the agents to move closer together. This increased closeness reduces the effective area coverage of the swarm.

Figure 5.13 shows a screen shot of the simulator using the basic-count algorithm. The reduction in GPS usage is limited to edge based agents of the swarm. Some perimeter agents are not identified as a coordinator due to the number of neighbours they have being ≥ 5. These agents tend to be indented on a perimeter edge. This feature will be discussed in more detail in chapter 6.

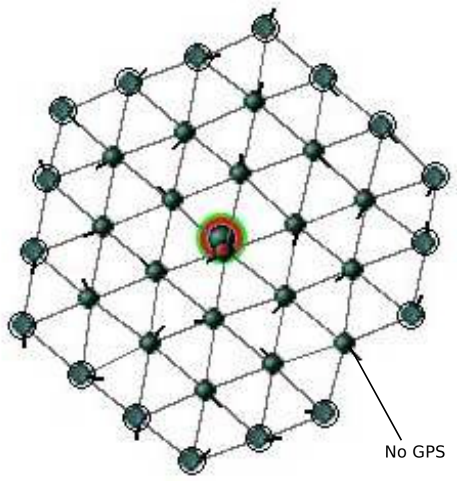
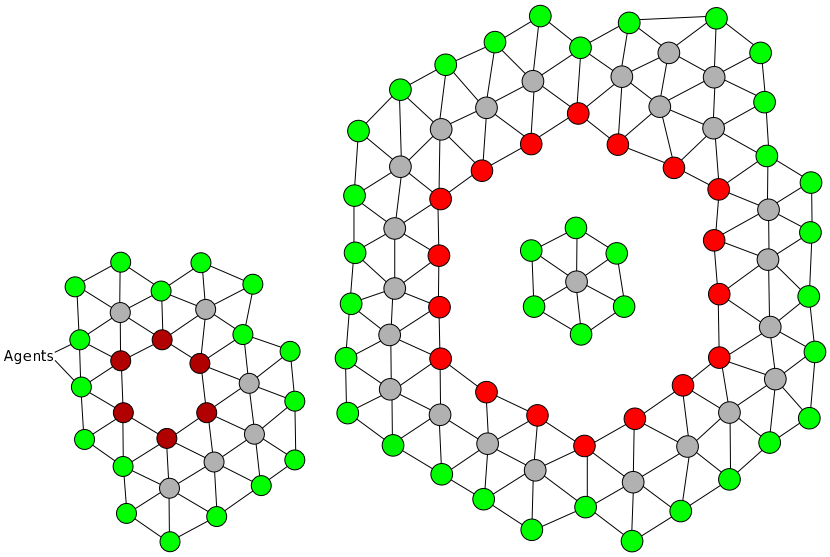


Fig. 5.13: Simple multifaceted agents (*screen shot from simulator*)

5.7 Complex multifaceted swarm (full-perimeter)

A complex multifaceted swarm is based upon the full detection of perimeters in a swarm. There are two perimeters that can be detected, convex and concave [92, 67]. Convex perimeters enclose agents and concave perimeters create voids. Figures 5.14(a) and 5.14(b) show these two perimeter types. The concave perimeter is highlighted in red and the convex perimeter is highlighted in green



(a) Void and perimeter (b) Multiple perimeters

Fig. 5.14: Swarm perimeters and voids

There is an exceptional circumstance that must be considered in perimeter detection. This is covered in § 5.7.1.1. Figure 5.15 is a screen shot from the simulator showing both perimeter types detected. The algorithm does not distinguished between the types.

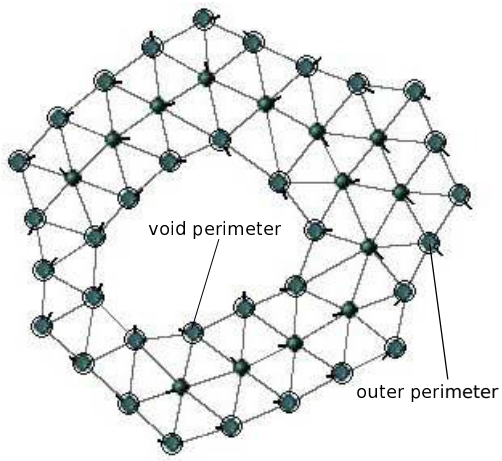


Fig. 5.15: Swarm with full perimeter detection (*screen shot from simulator*)

The full-perimeter algorithm allows a predictable subset of agents to be identified. This subset can be used to influence the overall direction of the swarm. The full-perimeter

detection algorithm detects similar agents to the basic-count algorithm but also identifies 'indented' agents that the basic-count algorithm ignores. The basic-count algorithm identifies these agents as having a neighbour count beyond the threshold. These additionally detected agents are used in void reduction which is discussed in chapter 6.

5.7.1 Full-perimeter coordinator detection

The process of detecting a perimeter agent is based upon identifying when the agent in not surrounded by interconnected neighbours (Figure 5.16). The process has several conditional checks that detect the status of an agent, each step is discussed below.

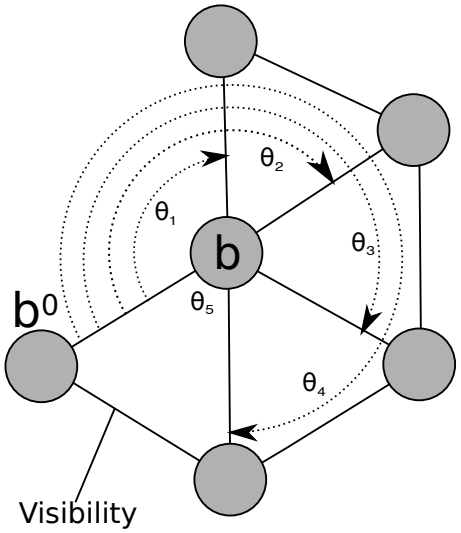


Fig. 5.16: Complex multifaceted agents

The first check is a short circuit count of the number of neighbours. This is the same process used to the basic-count algorithm. If an agent has ≤ 4 neighbours and the swarm parameters are for a hexagonal swarm configuration then the agent is a perimeter agent

and therefore a coordinator. This identification process is shown in figure 2.

Algorithm 2: SmallNeighbourCount

Data: b

`/* for` $nbr()$ `see Equation 2.1 on page 15 */`

if $|nbr(b)| \leq 4$ **then**

  return *True*

end

return *False*

If the short circuit neighbour count check fails then further conditions must be checked. The additional checks require a dictionary of agent/angular values to be generated (Equation 5.1). Algorithm 3 shows how the dictionary of agent/angle values are produced.

$$S_b \triangleq \{(b', \angle\{(b' \ b \ b^0)) : b' \in nbr(b)\} \tag{5.1}$$

Equation 5.1 generates the dictionary set of all the neighbours along with an angle that each of the agent's neighbour's make with the first detected neighbour.

Algorithm 3 shows the logic to produce the sorted neighbour/angle dictionary. $sort(.)$ sorts the dictionary set generated in equation 5.1 by ascending angle to produce a dictionary of neighbour agents with their relative angle to the first detected neighbour as

shown in equation 5.16.

Algorithm 3: NeighbourAngle: Sorted by angle

Data: b

`/* angles is a dictionary of agent/angle */`

$angles \leftarrow \emptyset$

for $b^{'} \in b.neighbours$ **do**

  if $b.neighbours[0] == b^{'}$ **then**

    $angles[b^{'}] = 0$

  else

    $newAngle = b.getAngle(b.neighbours[0], b^{'})$

    $angles[b^{'}] = newAngle$

  end

end

return $sort(angles)$

Using the angular dictionary from algorithm 3 it is possible to identify if adjacent neighbours can detect each other. If the agent is within all of its neighbours boundary then it is not on the perimeter (Algorithm 4).

As the agents are monolithic all agents have the same field effects the 'visibility' of two neighbours can be determined. The angle and distance of each neighbour pair and the angle they create allows cosine rule to be used to calculate the distance the neighbours are apart as shown in Equations 5.2 and 5.17. The distance is then checked against the neighbour range field effect. If the distance is $\leq N_b$ where N_b is the *neighbour field* then the agent can assume the neighbours have 'sight' of each other.

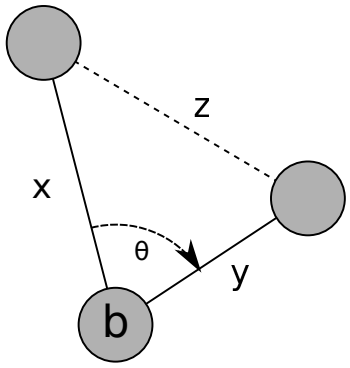


Fig. 5.17: Neighbour visibility

$$z = \sqrt{x^2 + y^2 - 2xy\ \cos(\theta)} \tag{5.2}$$

The full visibility check (Algorithm 4) takes each pair in turn and using the cosine rule (Equation 5.2) checks for a 'visibility gap'.

Algorithm 4: CheckVisibility

Data: b,angles

for $i \leftarrow 0$ **to** $size(angles) - 1$ **do**
    if *i == size(angles)-1* **then**
        if $cosrule(b, angles[size(angles) - 1][0], angles[i][0]$ **then**
            return *True*
        end
    else
        if $cosrule(b, angles[i + 1][0], angles[i][0]$ **then**
            return *True*
        end
    end
end
return *False*

There is one exception to the neighbour visibility check. Due to compression of the swarm, which can be caused by an initial deployment configuration (Figure 5.1 page 69) or when an obstacle is in the path of the swarm, the agent's neighbours are able to 'see' other neighbours but a pair of sequential neighbours could create an angle $> 180°$ as shown in Figure 5.18. In this case the agent is on the outside of the enclosed neighbour space, the agent is therefore on a perimeter.

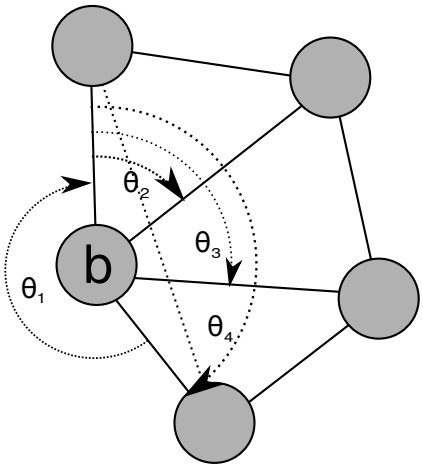


Fig. 5.18: Convex multifaceted agents

This phenomenon requires the detection process to identify the angles that all the agents create from an arbitrary point, in this case the first identified neighbour. The neighbours must then be checked to determine if any neighbour pair create an angle $> 180°$ (Figures 5.16 Algorithm 5). This process includes a short circuit in that as soon as a gap is found the check terminates.

Algorithm 5: CheckConvex

```
Data: angles
for i ← 0 to size(angles) − 1 do
    if i == size(angles)-1 then
        if 360 − angles[loop][1] ≥ 180 then
            return True
        end
    else
        if angles[loop + 1][1] − angles[loop][1] ≥ 180 then
            return True
        end
    end
end
return False
```

The complete perimeter detection algorithm using this 'cyclic-angular-neighbour-check' methodology is show in algorithm 6. The first part of the algorithm is the short circuit check (*SmallNeighbourCount(b)*). This is only followed by the visibility check (*Check-*

Visibility(b,angles)) if the short circuit fails. If the visibility check is confirmed, which is another short circuit method, the convex check is carried out (CheckConvex(angles)). This sequence ensures the minimum computational overhead for the agent check.

Algorithm 6: CheckPerimeter

```
Data: b, S
if SmallNeighbourCount(b,S) then
    return True
end
angles = NeighbourAngle(b)
if CheckVisibility(b,angles) then
    if CheckConvex(angles) then
        return True
    end
end
return False
```

5.7.1.1 Perimeter detection errors

With the proximity of the agents needing to be hexagonally connected and the requirement to eliminate inter-agent communications to allow for arbitrary sized swarms, there is the possibility of a localised perimeter detection algorithm error. When a swarm is compressed a localised anomaly can arise where agents are in a hyper-connected structure on a perimeter. When this anomaly occurs the proposed algorithm will produce a false positive result for the affected agents.

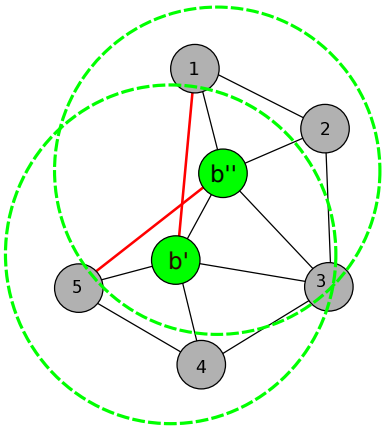


Fig. 5.19: Perimeter detection error

Figure 5.19 shows two agents $b^{'}$ and $b^{''}$. The agents have visibility of each other as shown by the green field effects. The agents can also 'see' another agent beyond their immediate neighbour. Agent $b^{'}$ is influenced by $b^{''}$ and neighbour 1. Agent $b^{'}$ is influenced by $b^{''}$ and neighbour 5. Due to these interactions both agents could calculate that they are surrounded by neighbours and are therefore not a part of the perimeter. The visibility issue is indicated by the red lines between $b^{''}$ and 5, and $b^{'}$ and 1. Neighbours 1 and 5 have no visibility of each other. Due to the connectivity shown in red there is a possibility of a failure in full perimeter detection as agents 5 and 1 cannot detect each other.

The possible pathways that could be detected as the perimeter are: $5 \rightarrow b^{'} \rightarrow 1$, $5 \rightarrow b^{''} \rightarrow 1$. A third alternative of $5 \rightarrow b^{'} \rightarrow b^{''} \rightarrow 1$ could also be used but it is not strictly a true perimeter route as the previous routes are shorter.

This problem can be resolved fully by introducing a communications channel that would allow the affected agents to negotiate a resolution to chose one of the shorter pathways [92], as there is no communications available this is not an option. In this thesis the error is limited to a very specific set of circumstances which will only occur during the initial expansion phase or during obstacle avoidance. The algorithm described in § 5.7.1 selects both agents as being perimeter agents therefore the problem is resolved by using the pseudo-perimeter ($5 \rightarrow b^{'} \rightarrow b^{''} \rightarrow 1$) ensuring a continuous perimeter is detected.

The screen shot below shows the algorithm resolving the issue to the pseudo-perimeter in the simulator (Figure 5.20).

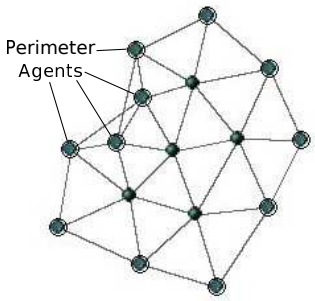


Fig. 5.20: Simulator perimeter detection

5.7.1.2 Perimeter and void detection

The focus of this thesis is the control of arbitrary sized swarms. The identification of the inner and outer perimeters would require a communications mechanism to determine which type of perimeter an agent was a part of [163, 92]. A mean angle $< 180°$ would indicate a void. A mean angle $> 180°$ would indicate an outer perimeter which may be enclose inside a swarm as shown in figure 5.14(b) on page 84.

The communications architecture would prevent swarms of an arbitrary size from being coordinated due to the $O(n^2)$ message propagation time factor (§ 5.10) [137]. It should be noted however that introducing the communications layer would increase the potential functionality of the swarm and may have practical applications in other scenarios where smaller swarms are appropriate.

A further issue of introducing a communications infrastructure would be the increase in the energy usage of the swarm. Identification of the perimeter type is therefore not practical in context of this thesis.

5.7.2 Baseline/full perimeter comparison

Figures 5.21 and 5.22 show the comparisons of the baseline swarm against the goal based swarm using full perimeter detection.

Figures 5.21 and 5.22 show that the settling period of the swarm from the disorganised state occurs at a very similar rate to the baseline but the swarm does not settle into a condition that has the same stability as the baseline. This deviation can be expected

as the algorithm's selection criteria is such that when the swarm is highly compressed there will be a limited number of perimeter agents effecting the directional bias of the swarm and the majority of internal logic will be the *interaction vectors* expanding the swarm.

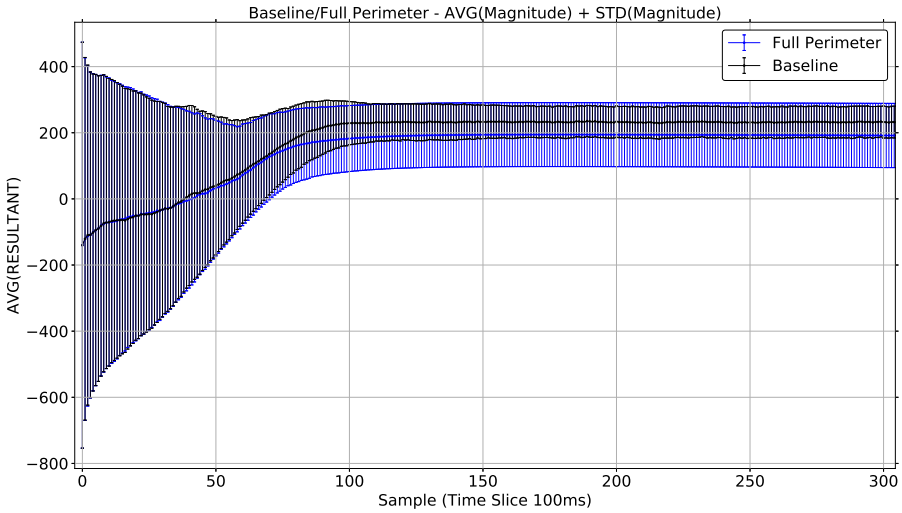


Fig. 5.21: Baseline/Full perimeter magnitude comparison

Figure 5.21 shows the magnitude metric for the simulation. Up to the point where the magnitude reaches zero ($\approx$ 4.8 seconds / 48 cycles) the swarm is undergoing expansion and the *inter-agent vectors* have a higher repulsive magnitude. Up to this point it is not possible to determine if the swarm is a cohesive entity. Once the *inter-agent vector magnitude* becomes positive the swarm is known to be cohesive. Once the *interaction vectors magnitudes* subside the directional bias from the coordinator agents *destination vectors* will influence the swarm's direction.

Once the swarm has completed its expansion the swarm's interacting vectors stabilise to an optimum level for the swarm environment parameters. The level of interaction between the vectors produces an increase in the jitter compared to the baseline. This change is induced by the coordinators *destination vectors*.

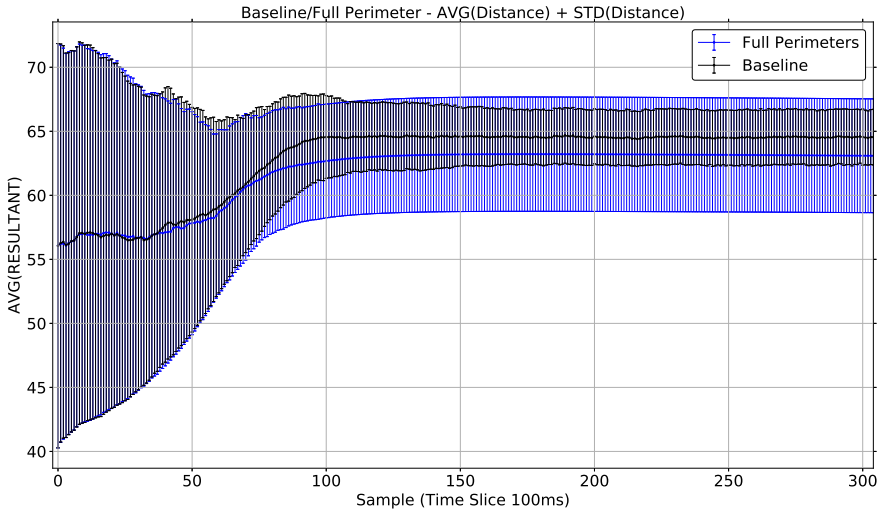


Fig. 5.22: Baseline/Full perimeter distance comparison

Figure 5.22 shows the swarm expanding as the average distance increases. The expansion eventually settles to a level where the agents are separated but the average distribution is less than the baseline. The reduction in the inter-agent distances is caused by a change induced by the *destination vectors* of the coordinators reducing the repulsion effect. The level of jitter is also greater than the baseline again this is caused by the *destination vectors* preventing the non-coordinator agents from moving towards an equilibrium state.

5.7.3 Complex Multifaceted Swarm (full-perimeter) - Simulation

The simulation shows that once the swarm has stabilised the majority of agents are still disrupted by the introduction of the *destination vectors*. This results in the swarm's agents having to move more to maintain the hexagonal structures. The agents that are on the perimeter tend to have 3 or more neighbours.

The experiment demonstrates that the perimeter detection algorithm is a practical technique to apply to the detection of a swarm subset to reduce GPS usage. However the environmental parameters the impact of the *destination vectors*create a high level of jitter within the swarm.

5.8 Baseline and effect comparison

To compare the three algorithms and their effects the metric defined in chapter 3 is combined with the swarm's emergent properties. The internal movement (jitter), the terminal speed of the centroid of the swarm and the effect on the path of the swarm are considered.

Analysing the path and speed of a swarm requires the centroid to be identified [53, 37, 39, 40, 38, 41, 42]. The centroid is then tracked over time as the swarm progresses towards it's goal (Equation 5.3).

The centroid of the swarm is determined by taking the coordinate position of each agent and calculating the mean of the x and y positions.

$$C_s = \frac{1}{|S|}\sum_{b' \in S} b' \tag{5.3}$$

Equation 5.3 calculates the coordinates for the centre of the swarm (C_s) as an x, y coordinate where S is the swarm and $b^{'}$ is the coordinate x, y for each agent and $|S|$ is the number of agents in the swarm.

Comparing swarms with these metrics generates a fuller understanding of the effects the algorithms and allows suitable applications to be identified.

5.8.1 Internal movement comparison

Figures 5.23, 5.24, 5.25, and 5.26 identify the jitter based on distance and the *inter-agent vector magnitudes* for the coordination algorithms and the baseline. Both the metrics show that as the number of coordinator agents increases the jitter increases.

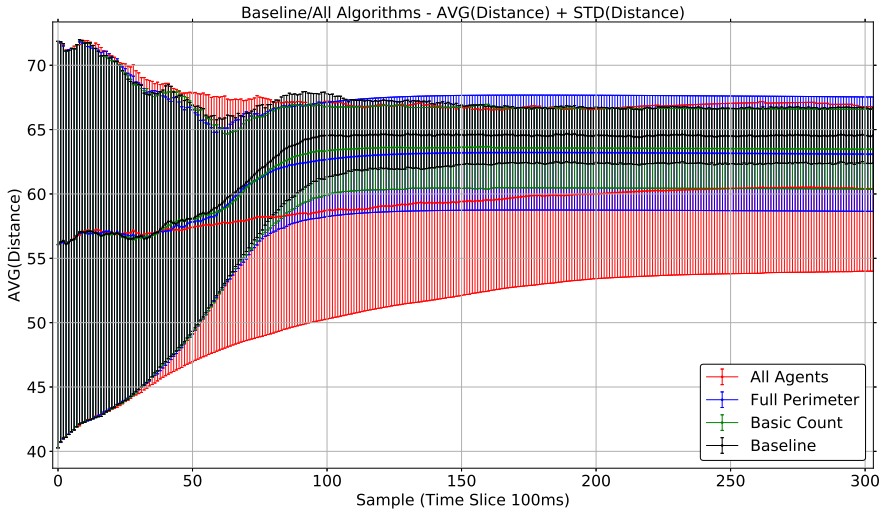


Fig. 5.23: Baseline distance comparison

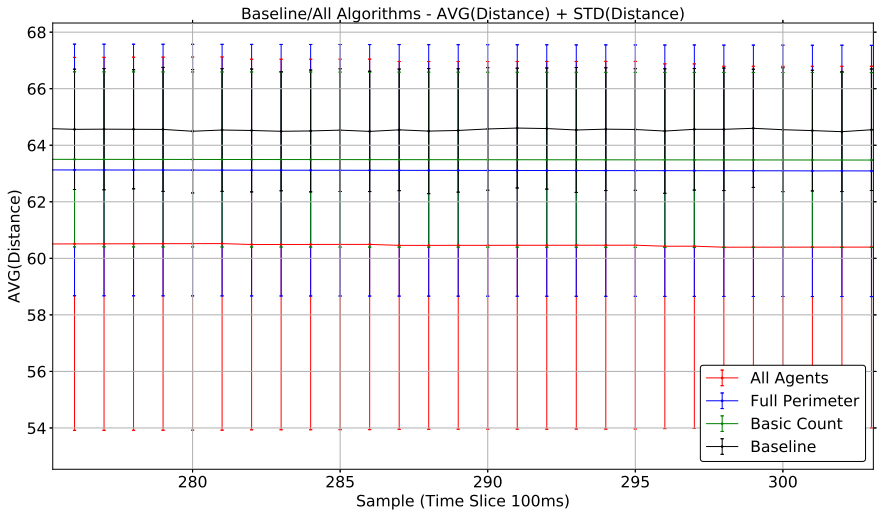


Fig. 5.24: Baseline distance comparison

Figure 5.24 show that the average distance between the agents is reduced when less coordinators are used. This indicates that for the same sized swarm with the same field effects the area covered by the swarm is reduced as the effect of the repulsion is

hampered by the coordinators. It also shows that the time taken for the swarm to stabilises is increased due to the *destination vectors* impacting on the agents ability to produce stable structures (Figure 5.23).

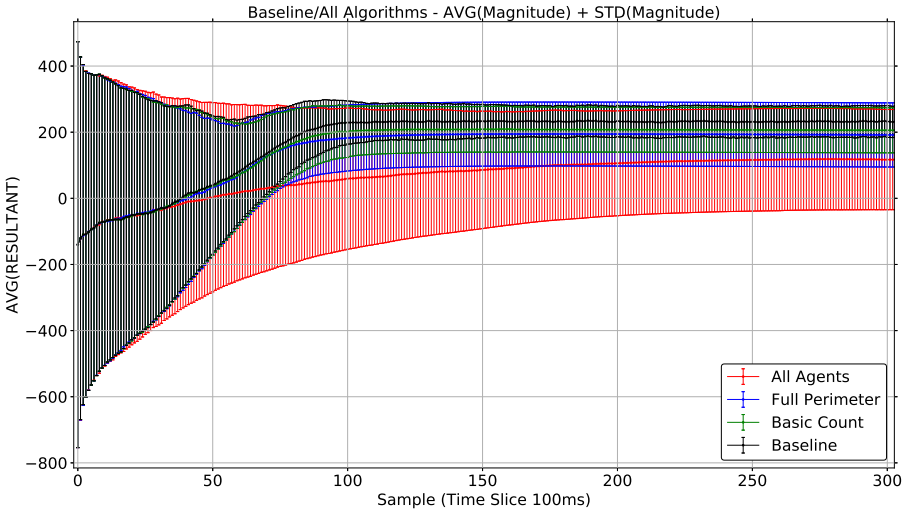


Fig. 5.25: Baseline magnitude comparison

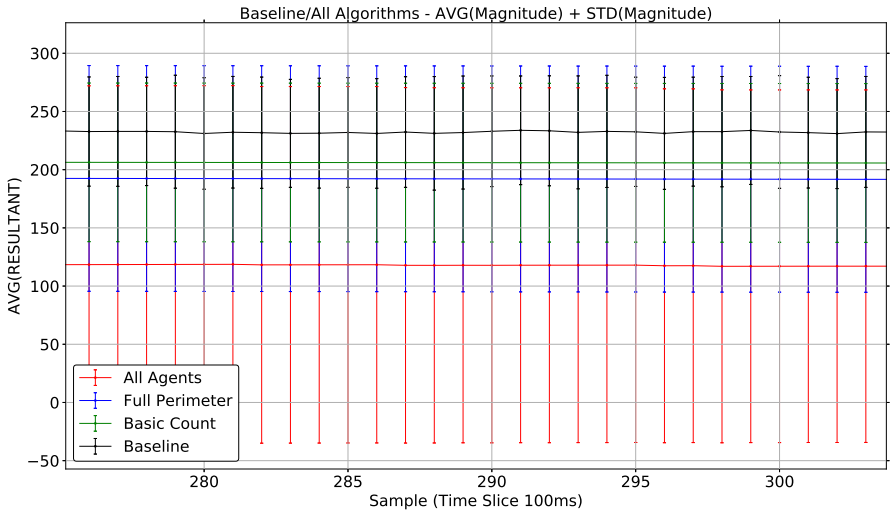


Fig. 5.26: Baseline and magnitude comparison

Figures 5.25 and 5.26 highlight that a highly compressed swarm's magnitude, indicated by a negative magnitude, will overwhelm most of a swarm's activity. This can be expected due to the size of the magnitudes that are affecting the swarm's movement. When all agents in a swarm are coordinators the directional effect can break through the highly cohesive stabilising vectors but this increases the level of jitter making the swarm's internal movement less predictable.

The directional algorithm that introduces the least amount of jitter is the basic-count algorithm. The basic-count algorithm is also the simplest of the multi-faceted algorithms to implement with a minimal computational overhead for the agents.

Figure 5.26 shows a closer view of the swarms following the stabilisation period between 17.5s and 20s. The basic-count algorithm produces the least disruption to the agents and has the highest average *inter-agent vector magnitude* indicating the swarm has a higher tendency to remain cohesive with the widest distribution. This effect is caused by the increase in the cohesion effect when the agents are more distant. This demonstrates that the algorithm produces a swarm that covers a greater area. The increased area coverage is confirmed by figure 5.24 which shown the greatest average inter-agent distance. The 'all-agent' algorithm has the smallest resultant *inter-agent vector magnitude* indicating that the swarm is more highly compressed (more repulsion is in operation). This is corroborated by the inter-agent distribution shown in figure 5.24.

5.8.2 Swarm GPS utilisation

The multi-faceted algorithms that are defined above create subsets of coordinator agents for the application of a *destination vector* bias of a swarm. These subsets can be identified to identify the GPS usage in the swarm at each time cycle (t). This allows a profile of the algorithms coordinator role identification process to be isolated to identify the level of GPS utilisation in the swarm.

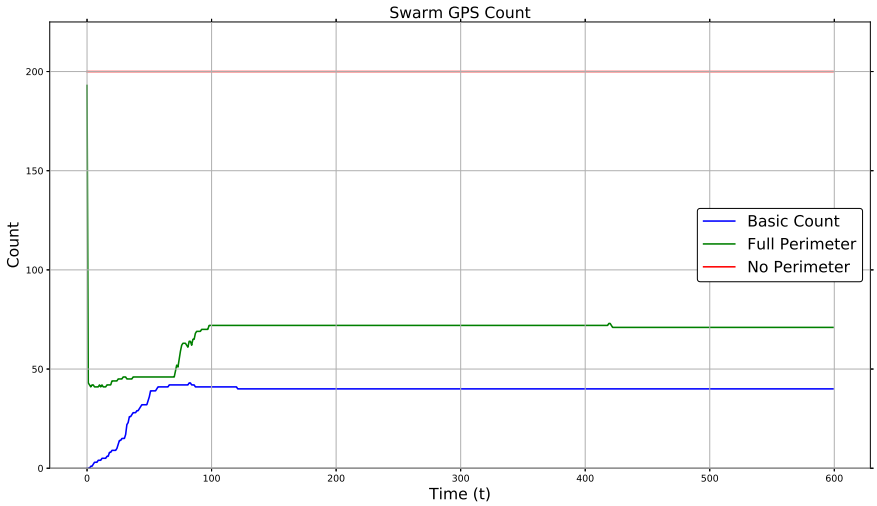


Fig. 5.27: 200 agent swarm GPS Usage

Figure 5.27 shows the development of the GPS utilisation of the swarm using the 'all-agent' coordination and the perimeter detection (multi-faceted) algorithms over the period of a 60s simulation.

5.8.2.1 All agent GPS utilisation

Figure 5.27 shows the 'all-agent' algorithm as a simple flat constant of 200 agents which is 100% of the swarm's agents. The directional bias is therefore constant throughout the simulation.

5.8.2.2 Basic-count GPS utilisation

In figure 5.27 the 'basic-count' algorithm is shown in blue. Due to the swarm being compressed the basic-count effect is initially limited. There are no coordinators identified at $t = 0$. This is due to the compression of the agents in the initial deployment. As the simulation progresses the disorganised stage expands the swarm and disperses the agents such that each agent develops less neighbours. The coordinator detection algorithm then begins to identify a subset on the perimeter. This process continues and eventually settles, shown by the plateau in the graph. Figure 5.27 shows there is a slight change in

the swarm structure at 8 seconds, this is due to anomalies in the swarm being corrected by the effect of the cohesion and repulsion field effects. Once the anomalies percolate out of the swarm the structure stabilises and the number of coordinators stops fluctuating.

5.8.2.3 Full perimeter GPS utilisation

The full perimeter detection algorithm starts with a subset immediately as shown in green in figure 5.27. This is due to the algorithm always detecting a full perimeter based on neighbour visibility. This has an immediate effect on the swarm as there is always a degree of *destination vector magnitude* influence on the swarm, even during the expansion phase. However, the *interaction vector magnitudes* are much larger than the *destination vector magnitude* so the progression towards the goal is hampered and the path of the swarm is erratic. The introduction of the *destination vector* influence from the very begin also disrupts the swarm's expansion and increases the duration of the disorganised period. As the swarm expands the algorithm detects more agents as coordinators as the number of perimeter agents increases.

5.8.3 Swarm path propagation comparison

All algorithms do eventually settle to a stable structure (for their environment parameters) with agents distributed in lattice structures. The time taken for each algorithm to achieve their stable distribution varies. This is due to the level of influence the directional bias has on the swarm. Table 5.2 shows the settled coordinator distributions at 12s into the simulation.

GPS Model	Bots GPS	% usage	Description
All-agents	200	100%	All agents are coordinators
Basic-count	40	20%	Minimal counting algorithm
Full perimeter	72	36%	Void detection and perimeter (capable of supporting void reduction)

Tab. 5.2: Swarm GPS enabled coordinators

The three algorithms take different amounts of time to stabilise (Figure 5.27) this has an impact on how long it takes each of the algorithms to impact the swarm's movement

towards the destination. This effect can be seen in figure 5.28 that shows the paths that each algorithm produces.

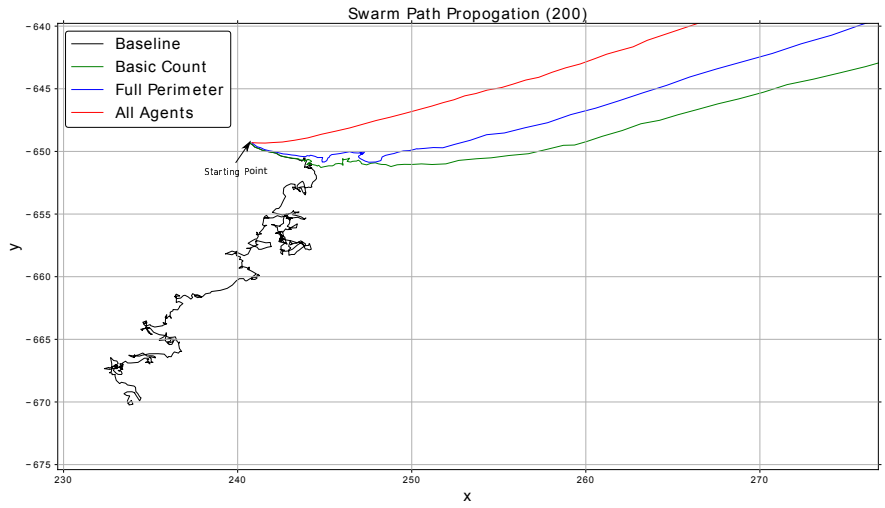


Fig. 5.28: Swarm propagation path comparison

Figure 5.28 shows the initial expansion phase of the swarm from deployment. Figure 5.29 shows the paths for the full simulation runs and the baseline comparison.

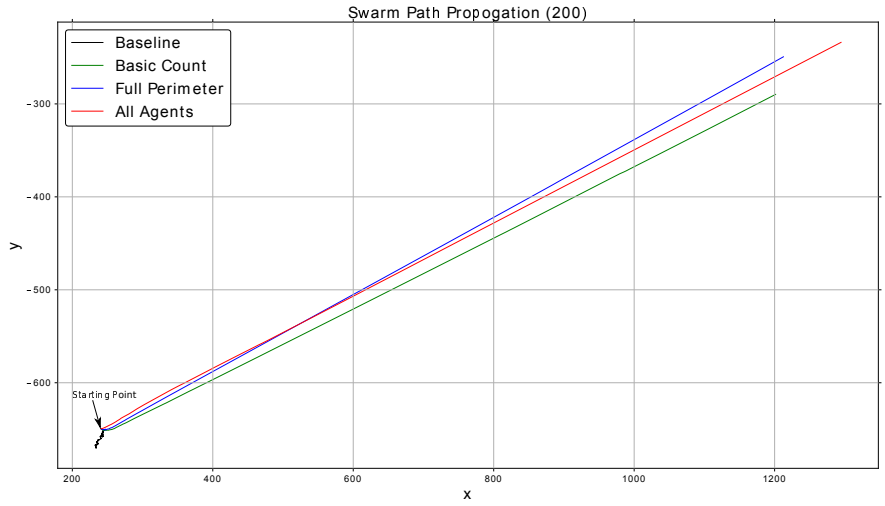


Fig. 5.29: Swarm propagation path comparison

Using all agents as coordinators results in the swarm immediately moving in the direction of the destination. The full perimeter detection has a limited effect on the direction of travel until the internal vectors start to balance and the internal magnitudes can be affected by a *destination vectors* at which point the swarm travels in the direction of the destination. The basic-count algorithm stabilises and then moves towards the destination in a similar manner to the full perimeter except in the initial compressed disorganised stage the basic-count algorithm develops in the same manner as the baseline swarm where there are no coordinators. These experimental results demonstrate that reducing the GPS utilisation has two effects on the swarm's propagation. Firstly it is possible to reduce the number of GPS sensors required to coordinate a swarm and it is possible to manage the 'jitter' while applying a directional bias on a swarm using *destination vectors*.

The effect of the algorithms on the stabilisation period can be seen in figure 5.30 which shows the swarm paths at the end of the 60 seconds simulation period. The relative positions shows the distances that the swarm is able to travel while employing the three algorithms, this is based on the swarm centroid. The baseline centroid path shows that without a *destination vector* being applied the swarm's centroid moves but it is based upon the *interaction vectors* only and there is no directional bias.

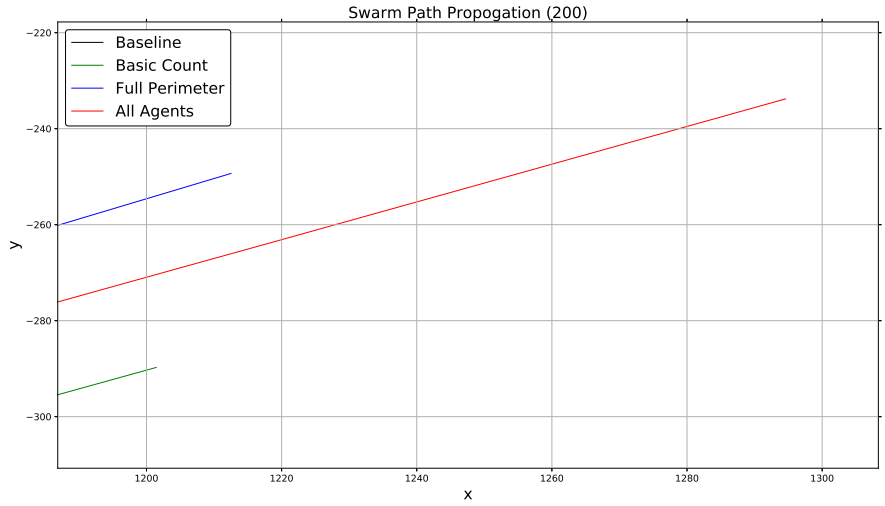


Fig. 5.30: Swarm traversal at end of 60s run

5.8.4 Speed of Swarm (Based on centroid)

To determine the speed a swarm is travelling a single reference point needs to be identified. A swarm's speed is measured based upon the position of its centroid, as discussed by Gazi and Passino [40, 39].

The speed at which the swarm travels is calculated as the distance the centroid moves (Equation 5.4) over a set time (Equation 5.5). Equation 5.4 shows the formulae for the distance a swarm has travelled over a set period of time, $t \rightarrow t^1$ where d is the distance the centroid has travelled, x, y is the centroid of the swarm and t is time.

$$d = \sqrt{(x_t - x_{t^1})^2 + (y_t - y_{t^1})^2} \tag{5.4}$$

$$S_s = \frac{d}{n} \tag{5.5}$$

Equation 5.5 shows how the speed is calculated based on the distance in equation 5.4. S_s is the speed of the swarm's centroid and n is the number of time increments.

Once the swarm has stabilised it migrates towards the destination. Each of the algorithms has an effect on how many coordinators are used to control the direction of the swarm (Figure 5.27). The impact of the number of coordinators is expected to affect the propagation speed of the swarm towards its destination. Based upon the final positions of the swarms over the last 20 seconds of their simulations it is shown that there was only a marginal impact on the speed of the swarm once stabilised (Figure 5.31). The difference in the algorithms only affects the time it takes the swarm to stabilise to the level that the *destination vectors* can influence the swarm's movement.

The results also show a reduction in energy usage (GPS usage) is possible with minimal effect on the overall speed of the swarm (taken from the centroid).

Although the agents within the swarm have travelled the same distances the overall path of the swarms has been affected (Table 5.4).

The time frame for the results in table 5.3, 5.4, 5.5 are for a time slice from 40 - 60 seconds in the simulation.

GPS Model	Start
All-agents	(922.5254184933, 379.965865509344)
Basic-count	(828.5991571583, 433.5750489635)
Full perimeter	(848.1131820415, 410.9739402922)

Tab. 5.3: Swarm centroid after stabilisation (40s)

GPS Model	End
All-agents	(1294.5940641812, 233.788749492987)
Basic-count	(1201.4534856023, 289.7373900729)
Full perimeter	(1218.1290381962, 260.3440891182)

Tab. 5.4: Swarm centroid after stabilisation (60s)

GPS Model	Distance	Speed
All-agents	399.7534569593	19.987672848
Basic-count	399.6368631077	19.9818431554
Full perimeter	399.5010461444	19.9750523072

Tab. 5.5: Swarm distance and speed after stabilisation (40s-60s)

GPS Model	End	Distance
All-agents	(1294.5940641812, 233.788749492987)	588.5813143506
Basic-count	(1201.4534856023, 289.7373900729)	696.0606134399
Full Perimeter	(1218.1290381962, 260.3440891182)	669.4489951522

Tab. 5.6: Distance to destination after run *(end = [1841,15])*

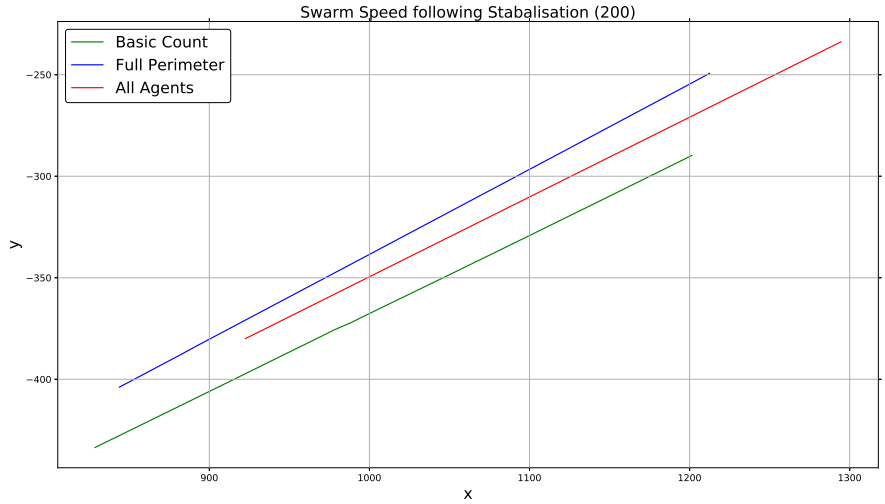


Fig. 5.31: 200 agent swarm path (over 20s period)

5.8.5 Alternate weightings for directional bias

The overall effect of the directional bias is effected by the number of coordinator agents in a swarm. Increasing the bias effects the jitter therefore balancing the weighting should have a positive effect in reducing the jitter.

The impact of the bias can be seen in the effect it has upon the number of neighbours an agent has during the path propagation.

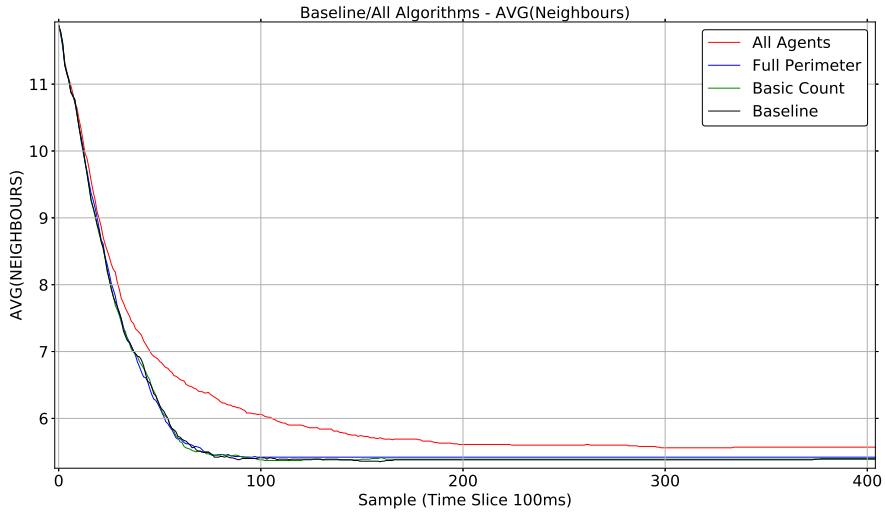


Fig. 5.32: Neighbour count effect from *destination vectors*

Figure 5.32 shows that for each of the algorithms the lattice effect (determined by neighbour count) takes longer to settle the more coordinators there are. In the case of the 'all agent' coordinated swarm the agents are affected most significantly and therefore exhibit the high variation in magnitudes and distances.

The coordinator statistics demonstrates (Table 5.2) the weighting of the *destination vectors* can be adjusted to produce an overall effect that is similar for each algorithm. If the weightings are adjusted taking the basic-count as the baseline goal-based swarm then the 'all agents' and 'full perimeter' algorithms can be weighted proportionally such that they have an overall *direction vector magnitude* that is similar to the basic-count table 5.7. The effects of this is shown in Figures 5.34, 5.36, and 5.33

GPS Model	Bots GPS	% Usage	Weighting
All-agents	200	100%	20
Basic-count	40	20%	100
Full perimeter	72	36%	60

Tab. 5.7: Swarm GPS proportional weighting

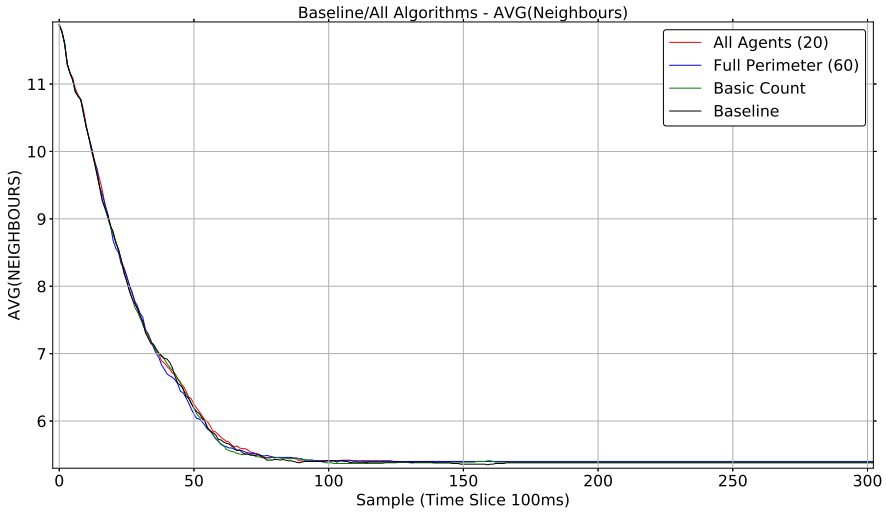


Fig. 5.33: Neighbour count effect from balanced directional bias

Figure 5.33 shows the effect reducing the *destination vector magnitude* has on the swarm structure. The reduction brings the stability (jitter) of the three algorithms in line such that all the algorithms have a similar impact on the swarm producing swarms with more stable structures while still creating a goal based affect.

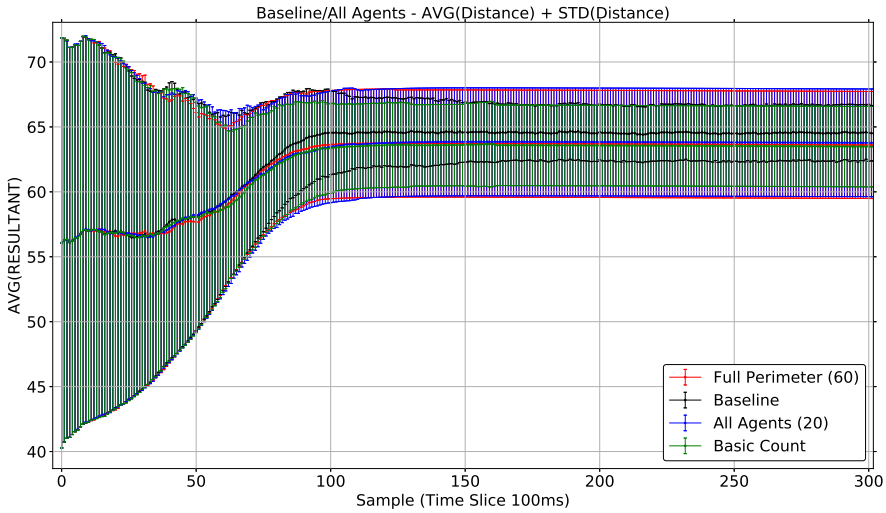


Fig. 5.34: Swarm distance analysis

Figure 5.34 shows that the stabilisation periods of all the algorithms are much closer when the bias is proportional and the disorganised stage for all the algorithms appear to be reduced to be within the same time frame as the baseline.

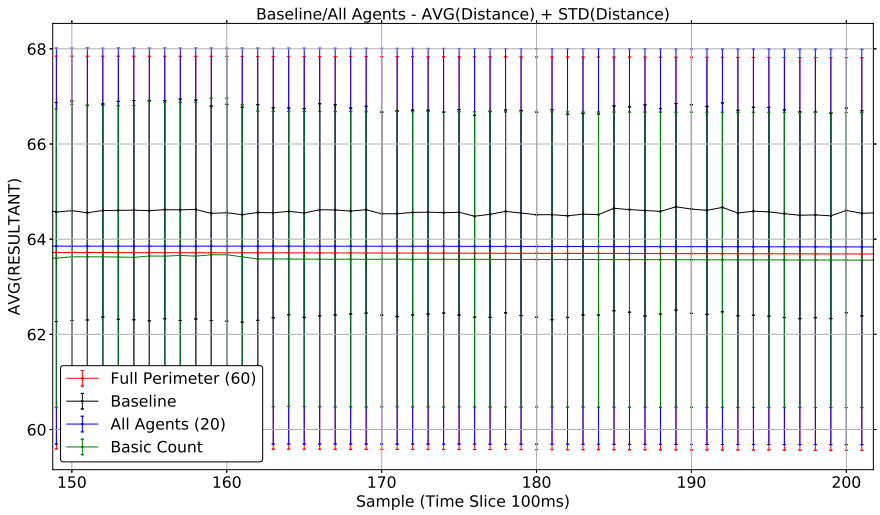


Fig. 5.35: Swarm distance analysis

Figure 5.35 shows that the resultant distance is also very close and the deviations are also very close. These results indicate that it is possible to reduce the internal jitter that is caused by an algorithm by adjusting the bias to weighting to be proportional to the number of agents that apply the directional influence.

Figures 5.36 and 5.37 show that the magnitude is also affected by the balancing of the directional bias. 5.36 shows the disorganised phase with the rapid expansion with the negative magnitude. This if followed by the stable period.

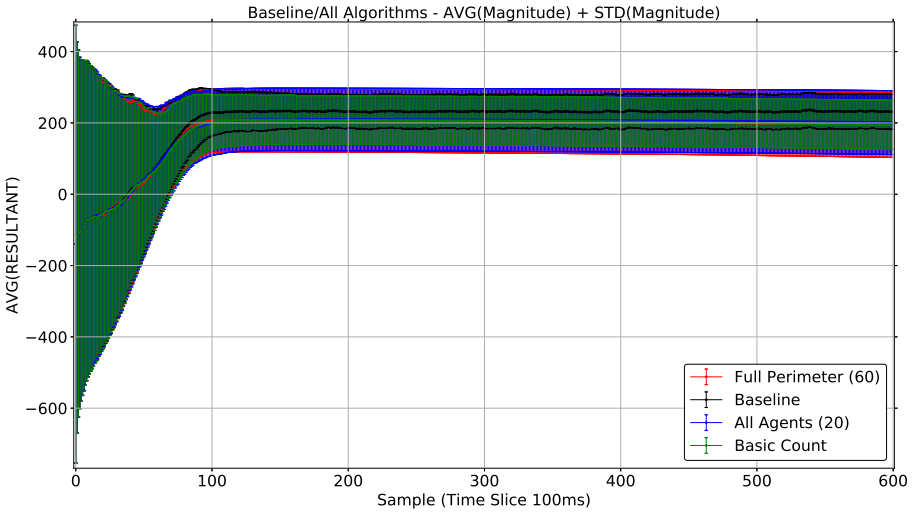


Fig. 5.36: Swarm magnitude analysis

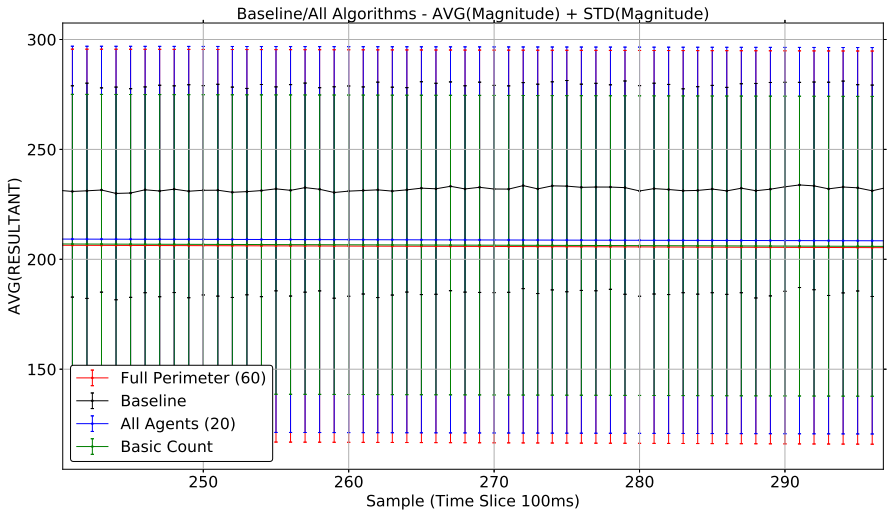


Fig. 5.37: Swarm magnitude analysis

Figure 5.37 shows the balance with the inter-agent magnitude for all the algorithms at a very similar level. It is noticeable however that the basic-count algorithm still has the lowest jitter.

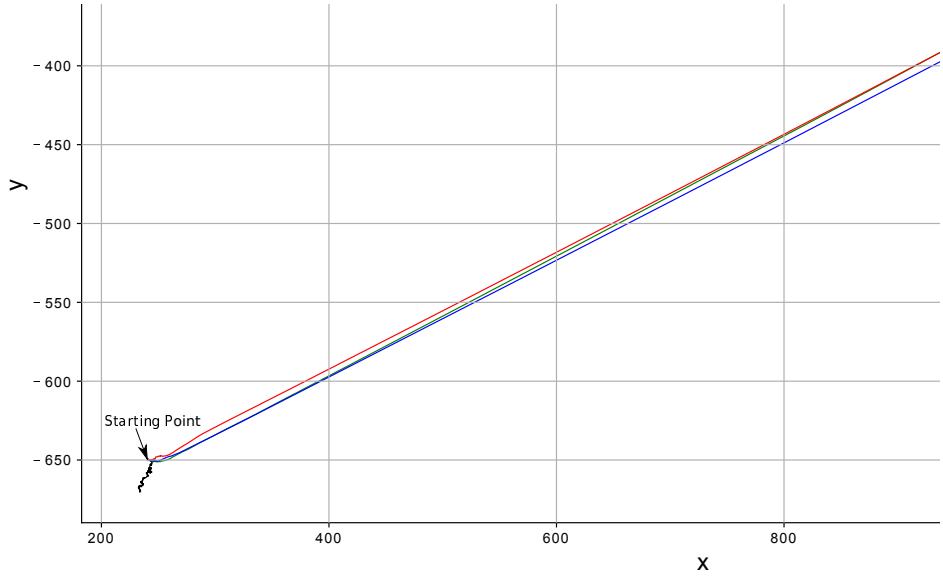


Fig. 5.38: Swarm path analysis

Figure 5.38 shows that with the adjusted weightings the swarms still progress towards their goal. Figure 5.39 shows that the reduced weightings impact on the directional path of the swarm and all the algorithms are effected by the *interaction vector magnitudes* during the initial expansion disrupting the swarm path.

Figure 5.40 shows that all the algorithms produce a goal-based swarm with the 'all-agent' algorithm furthest progress towards the end point. This is due to the instant directional influence of the algorithm.

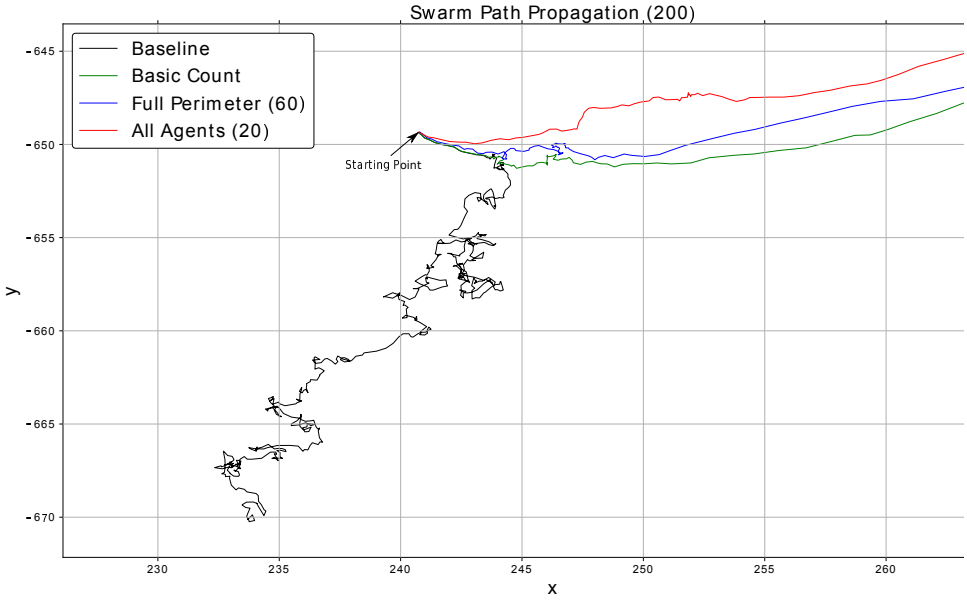


Fig. 5.39: Swarm path analysis

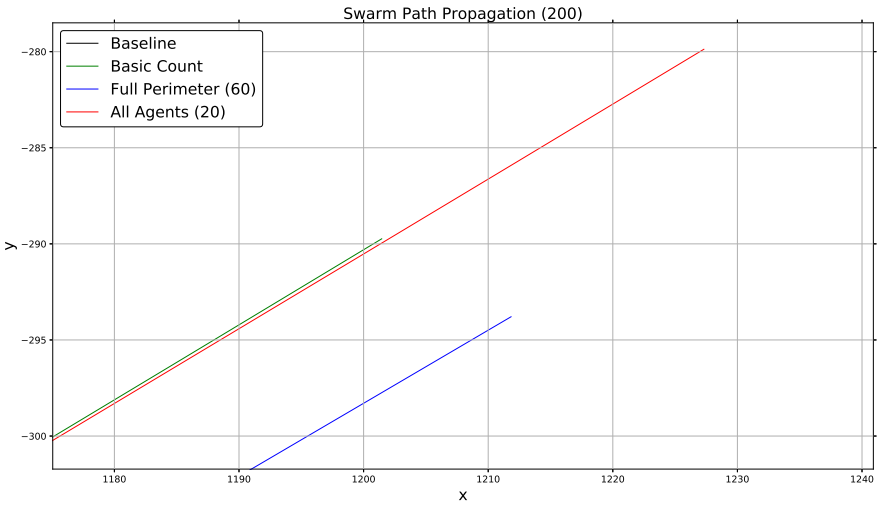


Fig. 5.40: Swarm path analysis

5.8.6 Swarm coordination evaluation

Comparing the three metrics it is possible to determine the coordination effects based on the requirements of the swarm.

The resultant terminal speed of the swarm, based on the centroid, is minimally effected by the algorithm.

The ability of the swarm to propagate towards a destination is hampered by reducing the number of coordinators as the stabilisation of the swarm's structure reduces the *destination vector magnitude*. The most effected algorithm is the basic-count algorithm which is unable to influence the swarm at deployment due to compression reducing the number of coordinators and lessening the aggregate *destination vector magnitude*. The full perimeter algorithm creates a greater aggregate *destination vector* more quickly than the basic-count algorithm, however the algorithm does induce additional jitter. The 'all-agent' algorithm has an immediate aggregate *destination vector magnitude* but this algorithm also induces jitter such that the hexagonal lattice start to fail.

Increasing the number of coordinators increases the amount of internal disturbance within the swarm which can affect a sensor array in terms of its ability to function efficiently. Adjusting the weighting of k_d can improve this as discussed in § 5.8.5. The algorithm that is most improved by adjusting the weightings is the 'all-agent' algorithm such that it allows the hexagonal lattice to form and still produces an immediate directional bias through the aggregate *destination vectors*.

The computational overhead of each of the algorithms differs only in the logic that is required for swarm coordination. The is no computation overhead for the 'all-agents' algorithm. The computational requirements of the basic-count algorithm is a simple count of neighbours and the full perimeter requires sweeps of angles and visibility checks which may impact on processor speeds for implementation.

The main energy difference between the algorithms is the use of the GPS sensor which is a high energy consumption device. Therefore for general purpose coordination the basic-count algorithm provides the lowest GPS energy requirement and also minimal computational overhead.

5.9 Energy efficiency evaluation

The effect of an algorithm also effects the amount of gross energy that is used by the swarm in creating the directional effect. If we consider different size physical agents the overall effect of the algorithm can impact on the practical use of the swarm.

The following evaluation is based on analysing the motor and GPS usage and ignores the energy consumption of sensors and agent processor units.

Reducing the number of GPS modules being used at any point in time will impact the overall use of energy by the swarm. Comparing current GPS modules it was found that on average GPS modules consume approximately 44mA, for a 3.3V GPS. This equates to approximately 145mW of power as given by Ohm's Law.

The energy usage of a motor will depend upon the power requirements required for agent movement (dependent on agent size and weight). It is assumed that the agents will utilise an omni-ball movement system to provide universal movement. Most omni ball systems utilise 3 motors to control the movement (Figure 5.41), other configuration are possible such as the four wheeled omni-directional wheelbase[118].

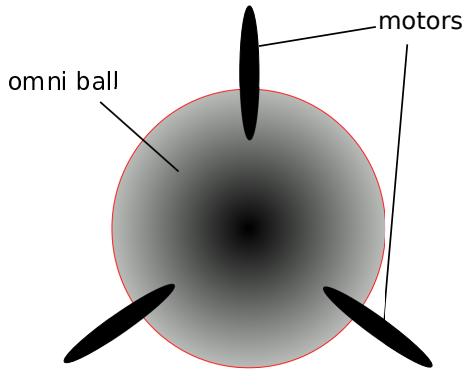


Fig. 5.41: Omni-ball motor arrangement

Three possible scenarios of different motor requirements are compared for possible energy savings.

The three scenarios are for 3A, 1A and 400mA motors.

The scenario assumes a GPS is being used in all the agents as shown in (Table 5.2) the energy consumption will be based on a swarm size of 200 agents.

Motor	Watts	No GPS	With GPS
100mA	0.33W	0.99W	1.1352W
1A	3.3W	9.90W	10.0452W
3A	9.9W	29.70W	29.8452W

Tab. 5.8: Energy consumption per agent

Motor	All 100/0	Full 36/64	Basic 20/80
100mA	227.04W	208.4544W	203.808W
1A	2009.04W	1990.4544W	1985.808W
3A	5969.04W	5950.4544W	5945.808W

Tab. 5.9: Energy consumption of swarm

Motor	All 100/0	Full 36/64	Basic 20/80
100mA	0%	8.186%	10.232%
1A	0%	0.925%	1.156%
3A	0%	0.311%	0.389%

Tab. 5.10: Energy consumption of swarm

Tables 5.8, 5.9 and 5.10 show the most effective savings in energy can be achieved by using smaller motors. This implies using smaller agents will provide the most appropriate platform to use GPS energy conservation. This efficiency saving falls in line with the current trend in using smaller agents in swarms as described by Mulgaonkar et al. [76, 99] who work as part of the research group at the University of Pennsylvania.

5.10 Message Propagation Performance

Many algorithms depend upon the propagation of messages throughout a swarm to determine many of the characteristics that allow it to be controlled [92]. However message propagation has an ($O(n^2)$) propagation profile which effects the speed at which a swarm can react to characteristic changes [137]. This could be countered by a reduction in the speed that agents move within a swarm to counter the message propagation delays.

The perimeter detection algorithms employed in this thesis have been reduced from $(O(n^2))$ to zero by limiting the agents information needs to simple proximity information from a sensor. This is a bio-inspired approach taken from the flocking of birds and shoaling of fish.

By reducing the message propagation to zero the scalability of the perimeter and void reduction algorithms are improved. Most research using message propagation techniques use swarms of at most 50 agents, as discussed this thesis is based upon the control of arbitrary sized swarms.

By localising the information requirements of an agent the storage requirements for message propagation are removed. Collating any characteristics of a swarm will require some form of local storage. As the messages propagate through a swarm more and more data is generated and data must be communicated and stored by each agent along with agent identifiers and some form of time stamp so data can be expired. In the case of the SenseSwarm algorithm this data is stored as a table of all agents, their coordinates, and a message id. This information is used to allows an agent to determine if it is at the lowest point in the swarm and therefore a perimeter agent [163].

5.10.1 SenseSwarm Message Propagation Comparison

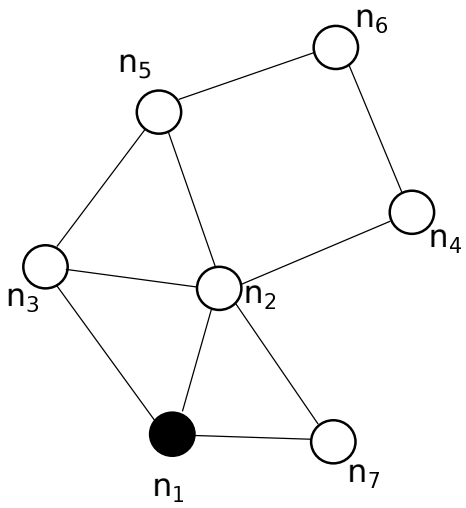


Fig. 5.42: Message Propagation

The SenseSwarm [163, 3, 4] algorithm detects perimeters using a mechanism by which a lowest point in the swarm is identified. This is achieved by each agent collating a table of every agents coordinates. Each message consists of an 'agent identifier' and a 'message id'. The message is constructed by an agent and forwarded to each of its neighbours. If a message is received with the same 'agent identifier' and 'message id' as a previous message it is discarded. If the message is detected for the first time then the message is forwarded to all of the neighbour-agent's neighbours.

for (n_1) the message propagation will be:-

$n_1 \rightarrow \{n_3, n_2, n_7\}$
$n_3 \rightarrow \{n_2, n_5\}$
$n_7 \rightarrow \{n_2\}$
$n_2 \rightarrow \{n_5, n_4\}$
$n_5 \rightarrow \{n_6\}$
$n_4 \rightarrow \{n_6\}$

The SenseSwarm algorithm works by locating the lowest agent in the swarm and then that agent propagating a message left and right to next lowest neighbours to identify the swarm's outer edge.

The SenseSwarm mechanism does not detect internal voids and is therefore not suitable for concave reduction as discussed in chapter 6.

5.11 Conclusion

This chapter discusses three techniques to create goal based swarms. The techniques are: Using all the agents to coordinate a swarm, using perimeter agents only and using a subset of the perimeter agents.

The techniques are then analysed using the metrics defined in chapter 3 to identify the effect the coordination techniques have upon the internal structures of the swarm.

Both metrics identify the state of a swarm with respect to variations in the disbursement of the agents from an average distribution and show that by reducing the number of coordinator agents it is possible to improve the internal distribution of agents within a swarm. The results showed that the effects the algorithms have upon the swarm's agents

could be balanced to reduce internal disturbances but that balancing the proportional effect would not impact on the energy usage of the positional sensors.

The chapter also discusses the removal of an internal communications architecture and introduces a proximity-based approach that allows for arbitrary sized swarms to be goal based.

6. SWARM COORDINATION - CONCAVE REDUCTION

Swarms consist of many agents and this is exploited to make them tolerant of failures. Failures occur for may reasons; for example: resource exhaustion, component failure, or a disruption from an external event. The loss of agents reduces the size of a swarm and changes the structure in places where the agents are lost. A swarm's structure can also be irregular from an initial deployment or due to an obstacle in its path.

These changes in the structure and/or size do not stop a swarm from functioning but can affect the outcome of a task to which the swarm is being applied.

The principle behind concave reduction is to counter the effect of agent loss or irregular structures within a swarm. Concave reduction is a technique where each agent identifies its position in relation to its neighbours and if the agent meets specific criteria it will attempt to move in such a way as to remove the gap between two of its neighbours. These changes in direction of multiple agents have a restructuring effect which results in a more uniform distribution of agents and reduces the perimeter size for the number of agents in the swarm.

When cohesion and repulsion are the only field effects creating a swarming effect (*inter-agent vectors*) the number of stable structures that can develop is limited. These structures are either straight edges or partial lattices. Partial lattices can create concave anomalies (dents) and convex anomalies (peaks) in a perimeter (Figure 6.1). An anomaly is any construct within the swarm that is not a hexagonal lattice or an agent distribution that causes the swarm to deviate from having only convex or flat edges.

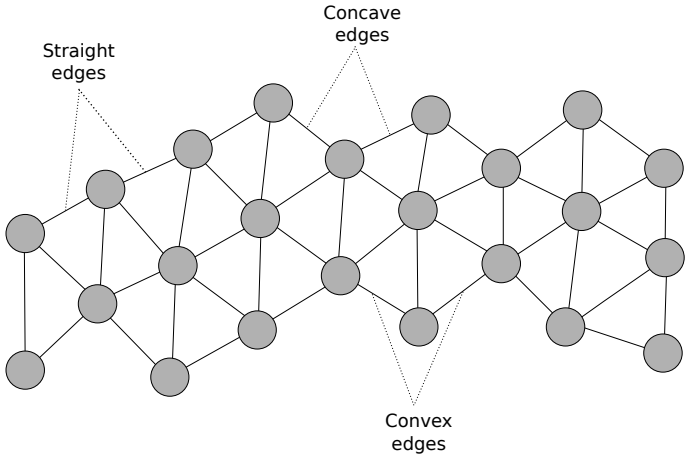


Fig. 6.1: Stable swarm edges

The problem with these anomalies is that although they are stable due to the *inter-agent vectors* they create non-uniform structures that may be unsuitable for a task that requires a generic swarm formation.

Concave reduction is a process that causes a swarm to coalesce towards a more generic shape. This is achieved by removing voids and concave edges to reshape the swarm to a structure with more hexagonal symmetry.

The technique defined in this thesis functions without the need for inter-agent or global messaging and is implemented at the local agent level through proximity detection.

Researchers have identified several approaches to resolving swarm structure issues. A prototype framework for self healing swarms was developed by Dai et al. [26] which considered the problem of agent failure in hostile environments. This is in line with work carried out by Vassev and Hinchey [153] who model swarm deployment using ASSL (Autonomic System Specification Language). This technique was used by NASA (National Aeronautics and Space Administration) when developing their ANTS (Autonomous Nano Technology Swarm) for use in asteroid belt exploration. This line of research is more focused towards an agent's internal systems failure rather than the removal of anomalies in a swarm distribution. Roach et al. [126] take a different approach and focus on the effects of sensor failure and the effect that has on agent distribution. The closest research to the concave reduction in this thesis is that of Lee and Chong [77] who identify the issue of concave edges within swarms in an attempt to create regular lattice formations. The main focus of their paper is the restructuring of the internal

distribution of inter-agent formations. The research by Ismail and Timmis [66] used the approach of *bio-inspired* healing using *granuloma formation*, a biological method for encapsulating an antigen.

Concave reduction is an extension of the work discussed by Ismail and Timmis in their paper on 'self-healing' [66] and Lee and Chong's work on identifying concave edges [77]. The technique also draws on the work of McLurkin and Demaine who have developed algorithms to detect perimeter types [92], although this thesis does not identify the perimeter types as this would require a communications infrastructure.

In a 3D space a sphere provides maximum volume with the minimum surface area and in a 2D Euclidean plane a circle provides maximum area with the minimum perimeter size. Both of these facts indicate that a perimeter should be tending towards a structure where its outer edges are convex in nature.

In chapter 5 perimeter detection is used to coordinate a swarm. Anomalies created by voids and dents add additional agents to the perimeter. This increases the size of a perimeter and reduces the efficiency of a swarm in terms of energy consumption when using sensors (GPS) for a goal based task.

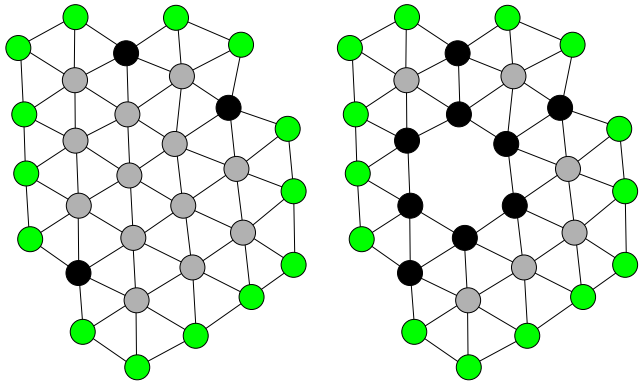
Using the full perimeter detection algorithm, as described in § 5.7, agents are identified as being perimeter based if they comply with a set of conditions. The conditions are:

- There are at least two adjacent neighbours that cannot detect each other (Section 5.16, page 85)
- The agent has less than 4 neighbours
- The angle between two neighbours is $> 180°$ (Figure 5.18, page 89)

These conditions create an identifiable set of adjacent agents (Figure 5.14, page 84) that create a perimeter.

This process detects both the outer edge of a swarm and any internal features (voids) that satisfy the same set of conditions. It is therefore possible to have both voids and islands of agents within the same swarm (Figure 5.14, page 84). Voids are best defined as perimeters that are concave in nature and inside another perimeter. McLurkin describes two types of perimeters convex and concave [92]. A convex perimeter is an edge where the average angle of the agent's exposed faces are $< 180°$. A concave perimeter is where the average exposed angle is $> 180°$ (5.14(a), page 84).

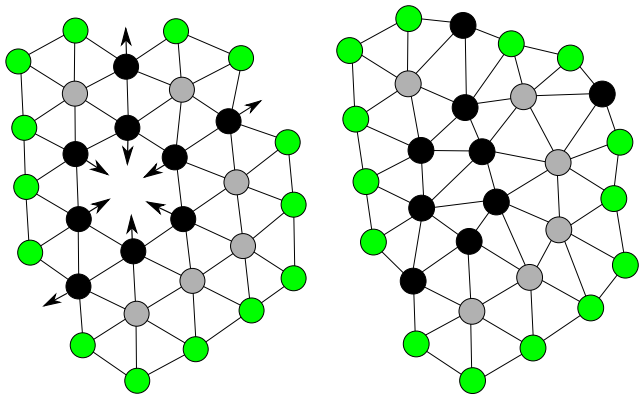
When discussing concave reduction the concave angle of interest is the relationship between an agent and two of its adjacent neighbours such that the neighbours have no visibility of each other and the neighbour-agent-neighbour relationship produces an angle $< 180°$. This is unrelated to the aggregate angle used to identify the perimeter type that the agent is a member of. Concave reduction therefore affects both inner (Figure 6.2(b)) and outer (Figure 6.2(a)) perimeter agents. Figure 6.2 shows candidate agents identified by this process in black.



(a) Outer Perimeter (b) Inner Perimeter

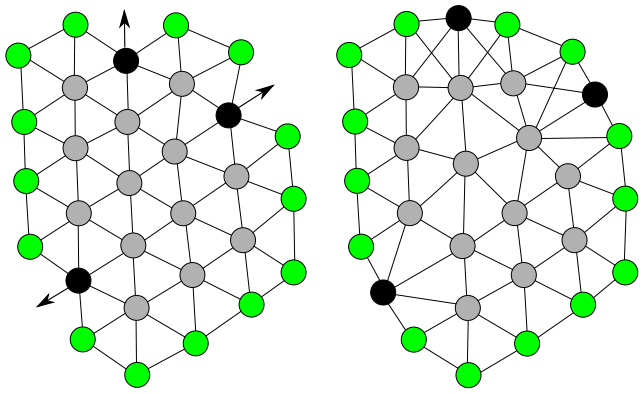
Fig. 6.2: Concave reduction agents

The purpose of concave reduction is to alter the overall structure of the swarm by replacing the calculated *movement-direction vector* on anomalous perimeter agents. These perimeter based changes cause a cascading movement within the swarm as non-perimeter based agent's *interaction vectors* attempting to move towards a position of equilibrium to their neighbours. The movement of the concave-perimeter (internal void) agents creates a tendency for the void to be removed (Figure 6.3) by percolating the void out of the swarm. On the outer perimeter deformities (indents) are straightened (Figure 6.4) and eventually a perimeter agent and its neighbours will tend towards creating a convex edge.



(a) Inner perimeter (b) Void reduction result

Fig. 6.3: Inner Perimeter Effects



(a) Outer perimeter (b) Void reduction result

Fig. 6.4: Outer Perimeter Effects

6.1 Concave reduction implementation

As previously shown in chapter 5 detection of a swarm's perimeter can be used to create efficiencies in the use of sensors to coordinate a swarm to a destination. Identifying agents that are suitable for concave reduction to be applied involves the same conditional checks as the perimeter detection algorithm with the addition of a status check on the cyclic-angular-check component (Figure 5.18, page 89). In a static swarm, where there are no *destination vectors*, concave reduction will result in a restructuring motion that creates a more 'rounded' swarm (Figure 6.4). Concave reduction also creates a surround effect as it removes voids from a swarm (§ 6.6). When concave reduction is applied to a

mobile swarm the void closing effect can be used to improve reconnaissance coverage as a swarm passes around obstacles (Figure 6.7).

Although these effects improve the potential applications of swarms there is a negative impact introduced by concave reduction. When a section of a swarm's perimeter has move concavities than its opposing side concave reduction can create an artificial aggregate *destination vector*. This aggregate vector causes the swarm to move towards the opposing edge.

To implement void reduction full perimeter detection (§ 5.7) is required. Concave reduction does not require the perimeter type to be identified and therefore no communications infrastructure is required [92, 104, 101, 165, 70]. The main limitation a communications infrastructure imposes on a swarm is the number of agents a swarm can contain. This is due to the propagation time of agent positions throughout the swarm as discussed in § 5.10 page 114.

The implementation of concave reduction identification is therefore incorporated into the perimeter detection algorithm (Algorithm 4, page 88). The changes are highlighted in algorithm 7.

Algorithm 7: CheckVisibility

Data: b,angles

$b.gap \leftarrow \emptyset$

$b.concave = False$

for $i \leftarrow 0$ **to** $size(angles) - 1$ **do**

  if i == *size(angles)-1* **then**

    if $cosrule(b, angles[size(angles) - 1][0], angles[i][0]$ **then**

      if $angle(b, angles[size(angles) - 1][0], angles[i][0]) < 180$ **then**

        $b.concave = True$

        $b.gap = (angles[size(angles) - 1][1], angles[i][1])$

        `/* Gap agents (first and last in the dictionary) */`

      end

      return *True*

    end

  else

    if $cosrule(b, angles[i + 1][0], angles[i][0]$ **then**

      if $angle(b, angles[i + 1][0], angles[i][0]) < 180$ **then**

        $b.concave = True$

        $b.gap = (angles[i + 1][1], angles[i][1])$

        `/* Gap agents */`

      end

      return *True*

    end

  end

end

return *False*

The purpose of the changes are to set a status flag to highlight a gap has been detected and to record the agent's neighbour pair that create the concave gap.

6.2 Concave reduction agent movement

Adding a further characteristic to the motion of a swarm necessitates a revision to the existing agent model as discussed in § 2.3 page 13. With concave reduction this revision

is based upon the identification of the concave perimeter edges as shown in figure 6.5 (b', b, b'').

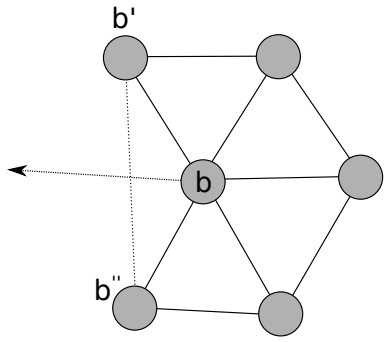
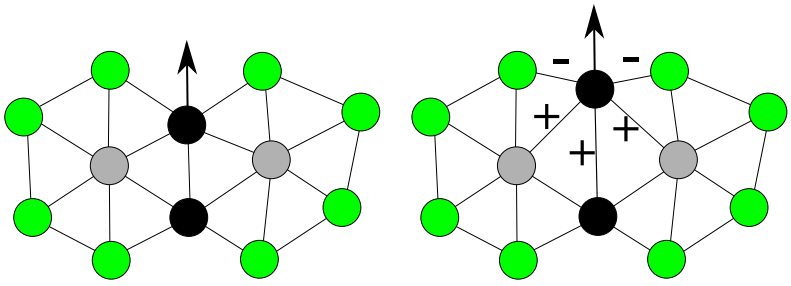


Fig. 6.5: Agent concave motion

When an agent is identified as being a component of concave characteristic (Algorithm 7) the normal *movement-direction vector* (Equation 2.8, page 21) is replaced by a *concavity reduction vector*. This new vector causes the agent to move in a direction that will reduce or remove a concave edge by moving the agent towards the identified gap. The effect of this will be to either straighten an outer perimeter or reduce/remove a void. This change in direction increases the distance and magnitude variances (jitter) due to the changes it will induce in the neighbours *interaction vectors*.

Figure 6.6 shows this effect in more detail. Figure 6.6(a) shows the initial positions of the agents before the concave reduction affects the agent. Figure 6.6(b) shows the positive and negative effect on the inter-agent distances that the movement creates. The aggregate change is an increase in the inter-agent distances.

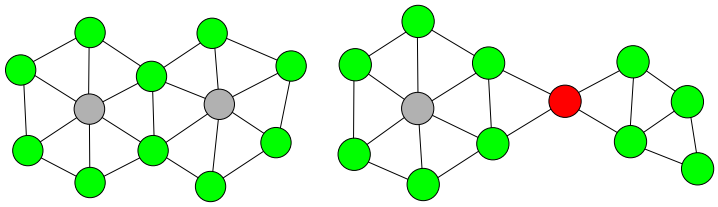


(a) Initial position (b) Reduced position

Fig. 6.6: Inter-agent effects

6.2.1 Perimeter Exceptions

When a swarm is forming or disrupted through either agent failures, catastrophic events or interactions with obstacles sections of the swarm may become isolated. When this occurs the sections can be considered as being a separate sub-swarms. It is also possible for sections of the swarm to remain connected through tenuous links of either 1 or 2 agents as shown in figure 6.7. In the case of a link connection (Figure 6.7(a)) concave reduction can be applied as normal. In the case of a 'bridge' link (Figure 6.7(b)) there is an issue in that there is more than one concave gap. As there is no communications infrastructure in swarm models there are limited options for handling this situation. One option is allow the swarming algorithm to apply as usual and no concave reduction takes place; the reasoning being that the overall structure of the swarm is unknown therefore the movement cannot be optimised. An alternative would be to select either the largest or the smallest gap; again as the structure of the swarm is unknown this may not be beneficial. A third option would be to select one of the gaps in some way (randomly, first seen, last seen) and implement the concave reduction at that point. The approach taken in this thesis is to select the first gap that is detected. Selecting the first gap reduces the computational overhead of the algorithm.



(a) Link (b) Bridge

Fig. 6.7: Perimeter Connectors

6.3 Concave reduction mathematical model

Equation 5.1 page 86 produces a set of agents sorted by angle and algorithm 7 page 124 produces a set ($b.gap \equiv G_b$) consisting of the first two agents identified as creating a 'gap' in agent b's neighbours. Equation 6.1 calculates the centroid of the gap agents.

$$D_{pos}(b) = \frac{1}{2} \sum_{b' \in G_b} b' \tag{6.1}$$

The centroid $D_{pos}(b)$ is then used to calculate the *concavity reduction vector* (Equation 6.2).

$$D(b) = D_{pos} b \tag{6.2}$$

$D(b)$ is a vector derived from the identified centroid of the neighbour gap (G_b) to the parent agent (b). This new vector is the *concavity vector* used to implement the concave reduction movement (Equation 6.2).

This derived movement is specifically for reducing the anomaly of which the agent is a part. The concave reduction movement needs to be enhanced to take into consideration an agent's surroundings. Due to proximity it can be assumed that there are no agents in the path between the agent and the gap, however there may be an obstacle. The *concavity vector* must include an obstacle avoidance component (Equation 6.3) as shown in § 2.6 on page 19. As with all the previous vector based calculations a weighting is applied to the calculated *concavity vector* k_{cr} to allow the model to vary the intensity of the effect. The resultant vector is then normalised to produce the *concavity reduction vector* as shown in equation 6.3.

$$V(b) = (k_{cr} D(b) + k_o v_o(b))\hat{} \tag{6.3}$$

6.4 Application of concave reduction on perimeter agents

If there is a 'gap' on a perimeter then the *concavity reduction vector* is applied instead of the normal *interaction vector*. To test this effect a baseline is established for a comparison.

Using the same swarm deployment as used in chapter 3 a baseline simulation is used as a comparison to identify the changes created by the introduction of the *concavity reduction vectors*. The comparison takes into consideration not just the jitter identified by the distance and magnitude metrics but also the effect on the number of perimeter agents. The main effect concave reduction on a swarm's structure is to reduce the size of the perimeter. Table 6.1 shows the simulation parameters for both the baseline and the concave reduction experiments.

Weight component	**Baseline swarm**	**Concave reduction**	**Description**
Sample rate	100	100	ms - Unit sampling interval
k_{cr}	0	100	weight adjuster for concavity reduction vector
k_c	5	5	weight adjuster for cohesion field
k_r	15	15	weight adjuster for repulsion field
k_d	0	0	weight adjuster for destination vector 0 for static baseline 100 from directional
Repulsion Boundary	70	70	units
Neighbour Distance	80	80	units
Speed	20	20	units/s

Tab. 6.1: Baseline comparison for concave reduction

Figure 6.8 shows the changes in the inter-agent distances that result from the concave reduction. The experiment demonstrates that with concave reduction the average distance increases and the variance of the distances increases. This is due to the 'pulling' effect of the *concavity reduction vector* on the perimeter. The algorithm distorts the distribution of the agents as discussed in § 6.2. The initial expansion of the swarm is similar for the baseline and the concave reduction however at 9 seconds into the simulation the *concavity reduction vectors* affects the swarm sufficiently to prevent the average distance reducing to the same level as the baseline. Once the swarm has stabilised with the concave reduction enabled the average distance and variance remain constant but above the baseline.

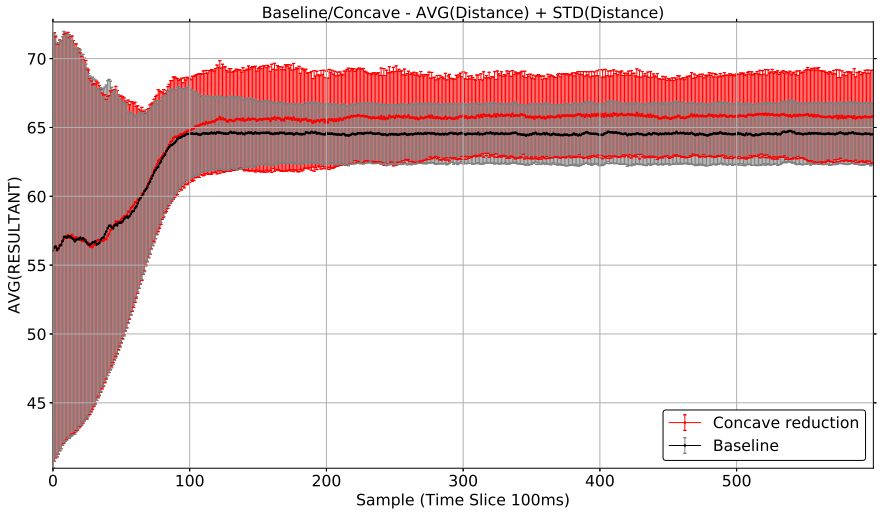


Fig. 6.8: Baseline/Concave effect distance

Figure 6.9 shows the change in the *inter-agent vector magnitudes* that the concave reduction introduces. Both the average magnitude increases and the variance. The magnitude increases as the *concavity reduction vector* has caused the agents to move outwards increasing the area of the swarm. The increase in cohesion is a direct result of the increased distance. The field effects still intersect but due to the average distance of the agents being further apart the repulsion is reduced. The agents being further apart causes the increase in cohesion as the algorithm attempts to prevents the swarm from breaking up. The variance increase is caused by the concave reduction effect moving selected agents into less optimal positions.

The *inter-agent vector magnitude* calculations in these experiments are based on the positions of the agents and the cohesion and repulsion field effects. The *concavity reduction vector* is not part of the metric calculation. The *concavity reduction vector* is only applied to an agent for movement calculation.

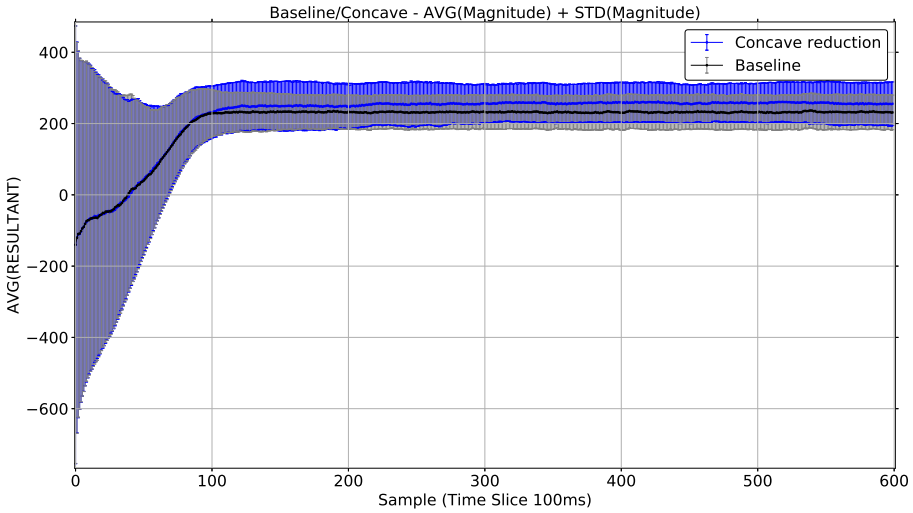


Fig. 6.9: Baseline/Concave effect magnitude

Figure 6.10 shows the effect on the number of perimeter agents. Initially the perimeter size is minimally affected due to the swarm's compression however after the expansion phase of the swarm (8 seconds) more perimeter agents are identified as concave edges are formed. As these concave edges are 'straightened' they create further anomalies that ripple through the swarm. This rippling is identified by the erratic changes in the number of perimeter agents. After approximately 30 seconds the swarm has been forced into a less angular structure with curved edges and the number of perimeter agents falls below the minimum of the baseline. The shape of the swarm is still undergoing change following this and the perimeter count continues to fluctuate.

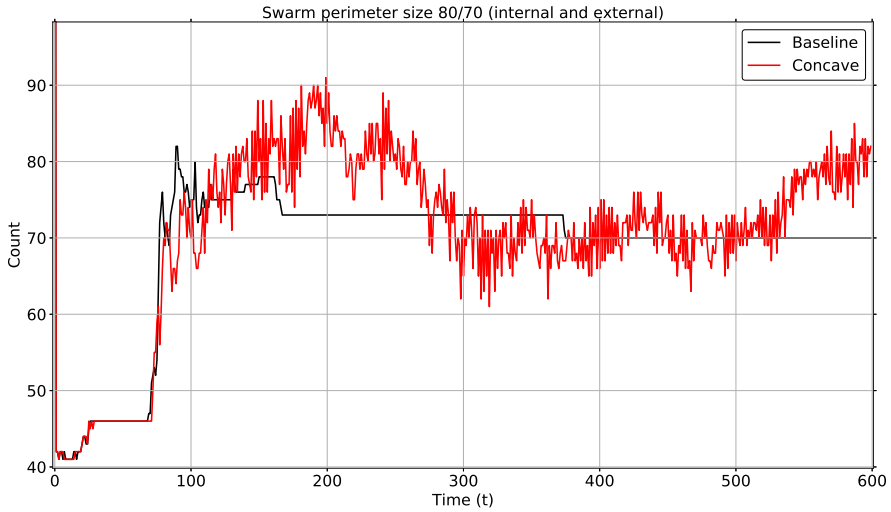
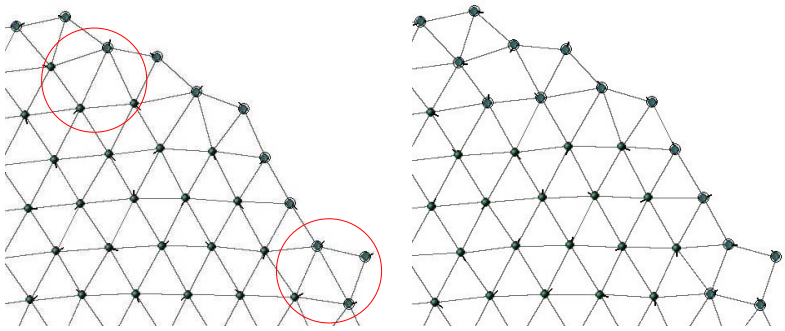


Fig. 6.10: Baseline/Concave perimeter size

The effect on the structure of the swarm caused by the concave perimeter agents moving towards the gap agents is to pull the internal agents forward. This pulling causes the swarm to develop a more rounded structure but the effect also have a negative impact. If the tolerance of the agent's movements (the difference between the agent ranges for repulsion and cohesion) is too small then the distortion effect of the concave reduction can create additional voids. The perimeter agents will then move so as to remove the defect (Figure 6.11(a)). Following the void creation the agents are now impacted by a second concave reduction from the newly created void and the anomaly on the perimeter edge 'snaps' back closing the void (Figure 6.11(b)). This process repeats itself creating an instability in the number of perimeter agents which is highlighted in figure 6.10 as the erratic change in the number of agents.



(a) Snapping effect closed (b) Snapping effect open

Fig. 6.11: Outer perimeter snapping effects

Figure 6.12 shows the paths of the agents in the swarm with concave reduction (red) and the baseline (black). The paths of the agents are initially very similar as the swarm expands but as the *interaction vector magnitudes* rise and the swarm stabilises the effect of the *concavity reduction vectors* start to noticeably influence the swarm structure. The most noticeable effect is on the perimeter where the agents have expanded then instead of stabilising to relatively stable fixed position there is drifting effect occurring due to the imbalance of the initial deployment structure (more anomalies on one side). The swarm also becomes more 'rounded' in appearance.

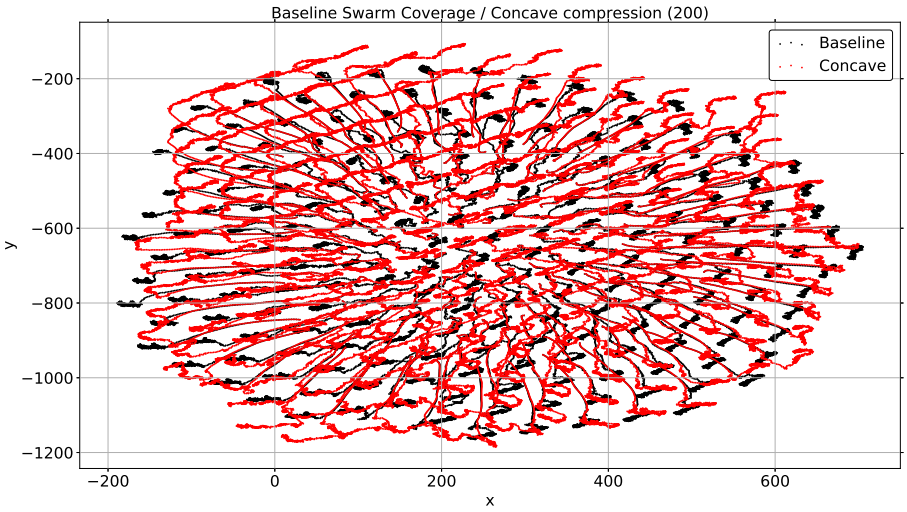


Fig. 6.12: Baseline/Concave path effect (after 600 iterations / 60s)

Figure 6.13 shows a more detailed view of the structure within the swarm. The baseline (black) paths show the swarm expanding and then settling to a hexagonal pattern which appears to oscillate slightly (jitter) when the swarm has reached its optimum distribution. The concave reduction swarm agents paths (red) show the swarm expanding in a similar way but once fully expanded the concave reduction causes the swarm to move with a slight directional bias.

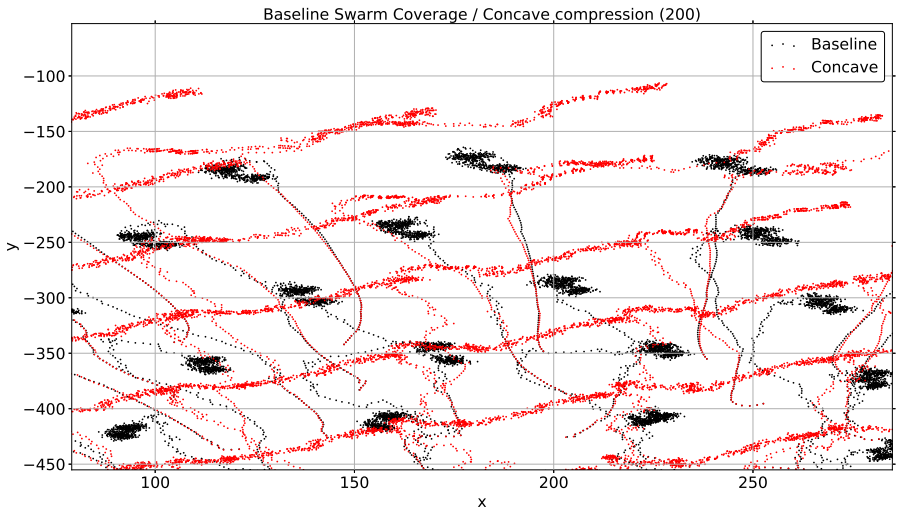


Fig. 6.13: Baseline/Concave path effect (after 600 iterations / 60s)

To reduce the 'snapping' effect (Figure 6.11), the field effect parameters can be adjusted to create a greater tolerance in the agent interactions. This can be achieved by either reducing the agents repulsion field and maintaining the neighbour field effect or increasing the neighbour field effect and maintaining the repulsion field effect. These changes affect the structure of the swarm but ensure that the agents to stay within the cohesion field when moving to implement the concave reduction the agents are therefore 'held' by the cohesion field effect preventing the 'snap'. These changes in the field effect can be shown experimentally using the parameters in table 6.2.

Weight component	**Baseline swarm**	**Concave reduction**	**Description**
Sample rate	100	100	ms - Unit sampling interval
k_{cr}	0	100	weight adjuster for concavity reduction vector
k_c	5	5	weight adjuster for cohesion field
k_r	15	15	weight adjuster for repulsion field
k_d	0	0	weight adjuster for destination vector 0 for static baseline 100 from directional
Repulsion Boundary	60	60	units
Neighbour Distance	80	80	units
Speed	20	20	units/s

Tab. 6.2: Baseline comparison for concave reduction

Figure 6.14 shows a comparison of the agent movements based on distance for the revised field effects. The graph shows that the swarm settles to a distance that is closer due to the reduced repulsion field. The graph also shows that the concave reduction still induces additional jitter as the variance is still greater than the baseline but due to the reduced snapping the perimeter agent count shows greater stability (Figure 6.16).

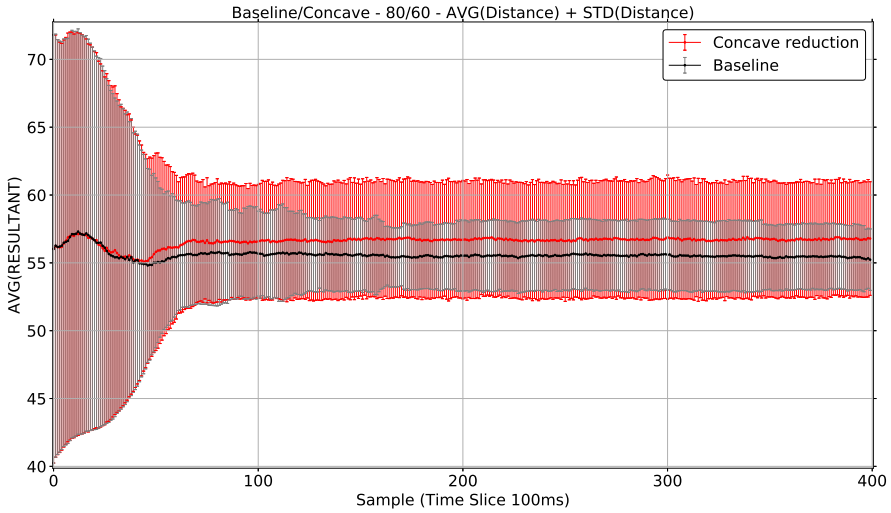


Fig. 6.14: Baseline/Concave effect distance (cohesion filed 80/ repulsion field 60)

Figure 6.15 shows that the *inter-agent vector magnitude* is reduced due to the reduced cohesion from the agent proximity and the *concavity reduction vector magnitude* impact is reduced due to less anomalies occurring on the perimeter.

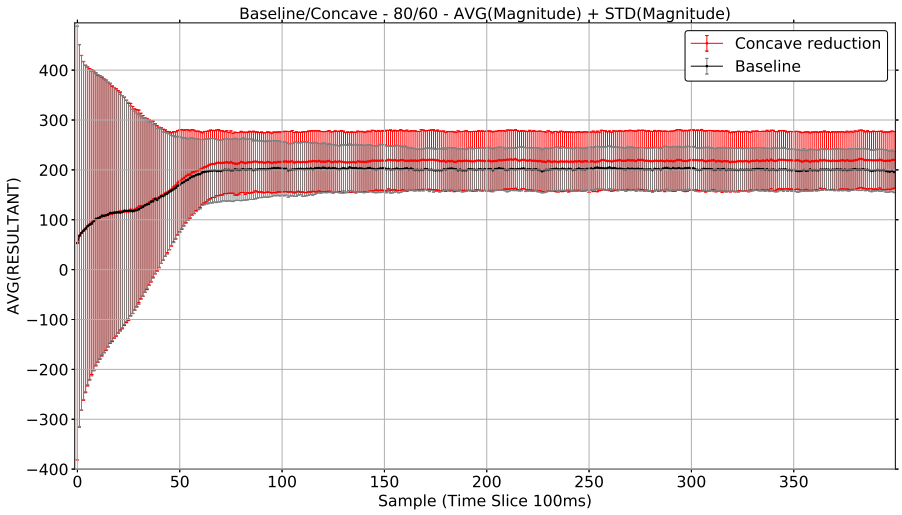


Fig. 6.15: Baseline/Concave effect magnitude (cohesion field 80 / repulsion field 60)

Figure 6.16 shows the changes in the perimeter size from the baseline and the concave reduction swarms. The concave reduction still creates an erratic perimeter count but the variation is reduced. On aggregate for the run the concave reduction has reduced the perimeter size. The revised effects also reduce the snapping effect and the swarm has an improved structure. The concave reduction creates a perimeter which fluctuates between 45-50 agents where as the baseline swarm settles to 49 agents.

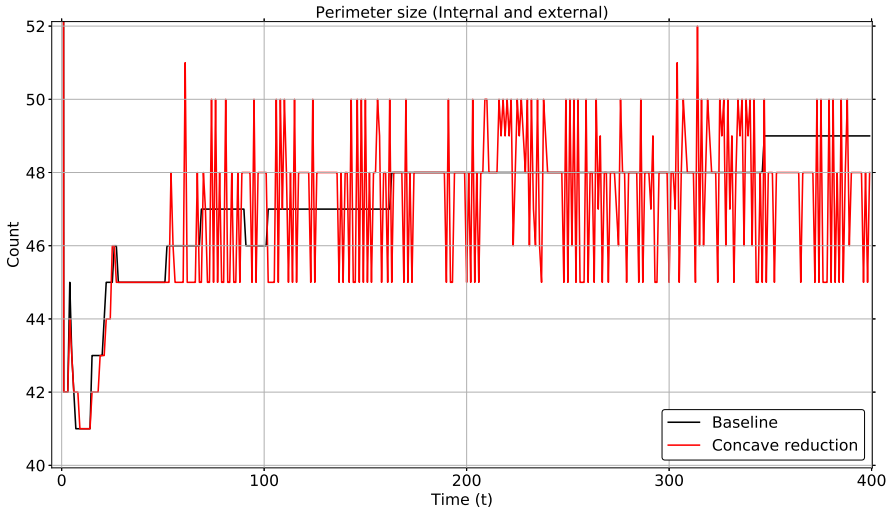
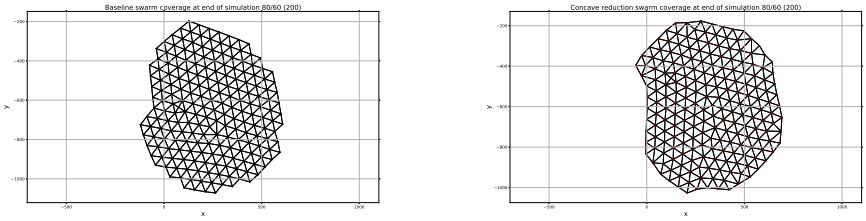


Fig. 6.16: Baseline/Concave perimeter size (cohesion field 80 / repulsion field 60)

For the period of the simulation the baseline had 19049 perimeter agents and the concave reduction had 18940 which is an improvement of 0.5% overall for the whole simulation (Table 6.3). This is only a small change in the perimeter size but the impact on the swarm structure is significant. The *concavity reduction vectors* have 'pulled' the swarm into a more circular shape (Figure 6.17).

Neighbour / minimum	**Baseline swarm**	**Concave reduction**
80/60	19049	18940
80/70	27394	27987

Tab. 6.3: Comparison of perimeter size



(a) Baseline end (b) Concave reduction end

Fig. 6.17: Simulation end points

Figure 6.18 shows the paths of the agents in the swarm with concave reduction (red) and without (black). The paths are similar as the swarm expands due to the *repulsion vectors* being large and masking the *concavity reduction vectors* due to the resultant *movement vector* directing the agents in a similar direction. Once the swarm has expanded the *concavity reduction vectors* create a more spherical appearance to the swarm. The *concavity reduction vector magnitudes* are significantly large enough now to create a slight directional bias as shown in both Figures 6.18 and 6.19. This effect is the result of the *concavity reduction vectors* pushing the anomalies on the left of the swarm resulting in the swarm moving slightly to the right due to the swarm having fewer anomalies on the opposite side of the swarm. The anomalies can be seen in figure 6.18.

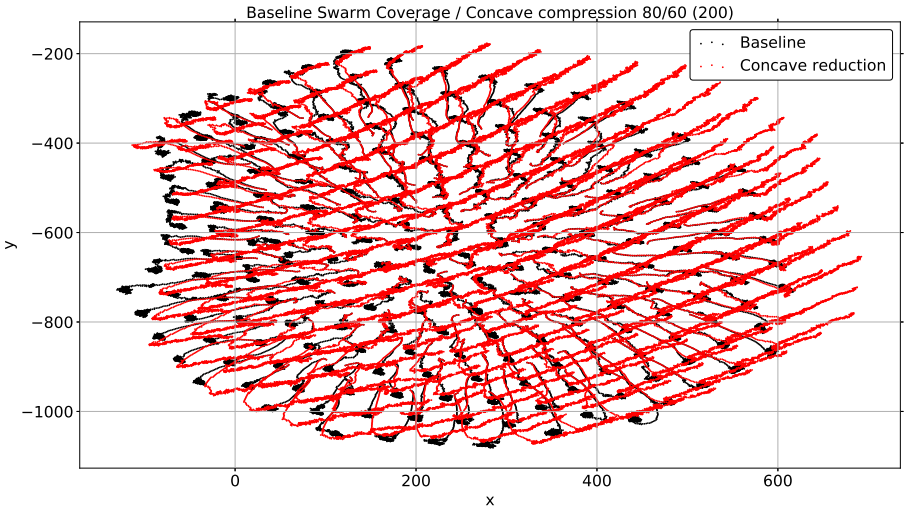


Fig. 6.18: Baseline/Concave path effect (repulsion field 80 / cohesion field 60)

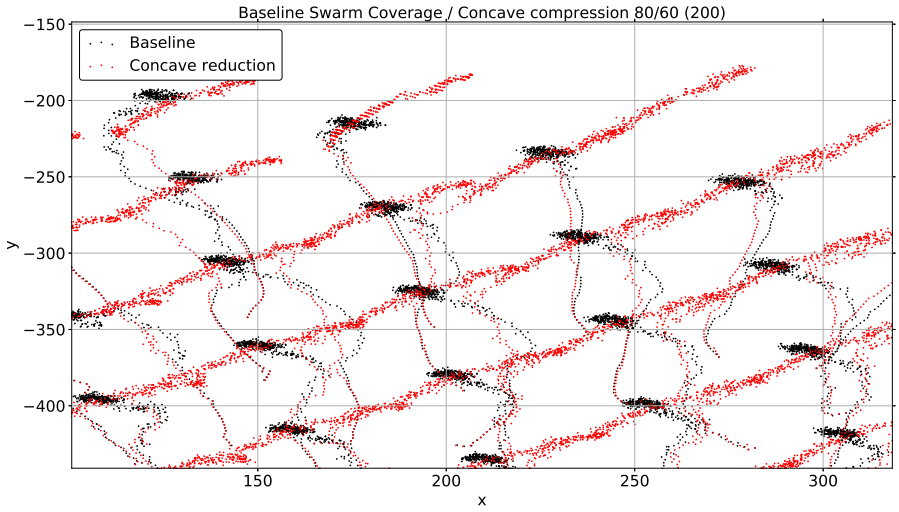


Fig. 6.19: Baseline/Concave path effect (cohesion field 80 / repulsion field 60)

With the tolerance levels set appropriately the impact of the concave reduction is to reduce the number of agents being identified as coordinators (Figure 6.7). This reduction in perimeter size is due to a reduction in the number of anomalies in the swarm. The effect of the concave reduction on a convex (outer) perimeter is limited when a swarm is deployed in an almost circular manner as there is limited space for optimisation. The effect is more pronounced when a swarm is 'malformed' with large anomalies producing concave edges.

6.5 Application of concave reduction on concave perimeters (voids)

Just as a gap on a convex perimeter is affected by concave reduction so is the perimeter of a concave perimeter (a void). The effect on a concave perimeter is more pronounced due to there being a higher ratio of concave anomalies. It is possible to have no concave gaps on a convex perimeter (a circular swarm) but a concave perimeter (void) always has concave anomalies. This characteristic allows voids to be controlled. To test the effect of concave reduction on void removal a baseline must be established (Table 6.4).

As with the previous testing of algorithm effects the comparison needs to take into

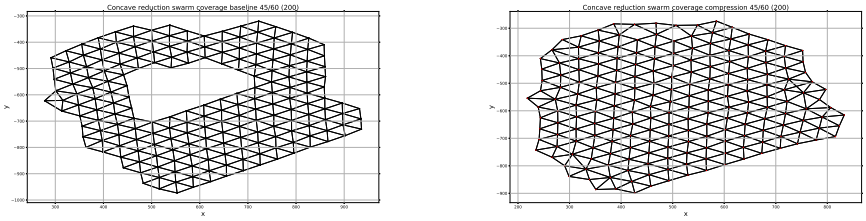
consideration jitter (inter-agent distance and *inter-agent magnitude*) and the effect on the number of coordinator agents (perimeter agents).

Figure 6.4 shows the swarm parameters for void removal by concave reduction.

Weight component	**Baseline swarm**	**Concave reduction**	**Description**
Sample rate	100	100	ms - Unit sampling interval
k_{cr}	0	100	weight adjuster for concavity reduction vector
k_c	5	5	weight adjuster for cohesion field
k_r	15	15	weight adjuster for repulsion field
k_d	0	0	weight adjuster for destination vector 0 for static baseline 100 from goal-based
Repulsion Boundary	45	45	units
Neighbour Distance	60	60	units
Speed	20	20	units/s

Tab. 6.4: Baseline comparison for concave reduction

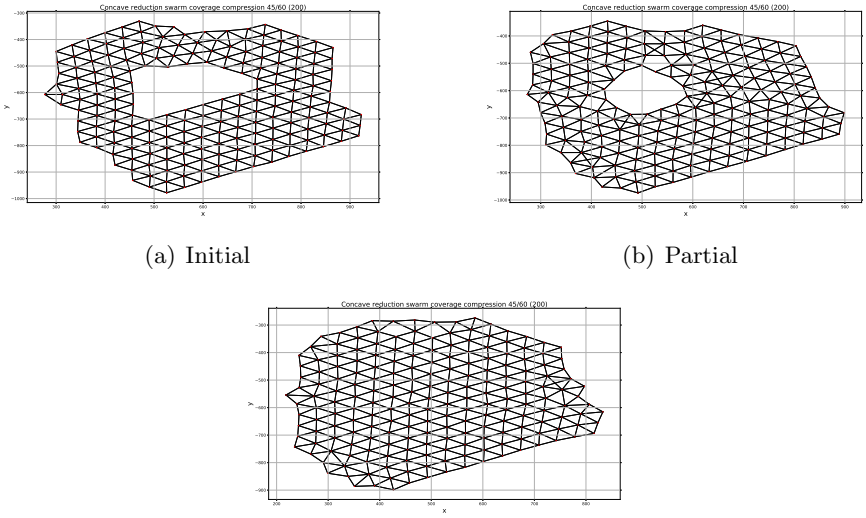
Figure 6.20 shows the end points for the simulations for the concave reduction experiments. Figure 6.20(a) shows the end point for the baseline. It shows that at the end of the simulation the void within the swarm persists. This is due to the distribution of the agents being optimal for the given simulation parameters and the structures within the swarm being 'stable'. Figure 6.20(b) shows the end point for the simulation with concave reduction enabled using the same swarm and configuration parameters. The concave reduction algorithm has removed the void from the swarm completely and the outer perimeter has been 'smoothed'.



(a) Baseline end (b) Concave reduction end

Fig. 6.20: Simulation end points

Figure 6.21 shows the stages that the swarm goes through when the concave reduction closes the void. The initial deployment of the baseline and concave reduction swarm are the same (Figure 6.21(a)). The concave reduction slowly reduces the void (Figure 6.21(b)) and smooths the outer perimeter. Once the smoothing begins the internal anomaly is filled from behind as the inner agents are drawn into the void. This causes small voids to 'percolate' outwards through the swarm until they meet an outer perimeter. Two of these 'percolating voids' can be seen in figure 6.21(b) to the left of the void and above the void. Once the 'percolation' process has completed the void is closed (Figure 6.21(c)). The swarm edges also take on a more 'rounded' appearance caused by the edges of the swarm being 'pulled' to create a convex edge.



(a) Initial (b) Partial

(c) Complete

Fig. 6.21: 200 agent swarm with void

Figures 6.22, 6.23, 6.24, 6.25 show the effect the concave reduction has on the inter-agent distances and *inter-agent vector magnitudes* for the simulation.

Figure 6.22 shows that the process of reducing the void increases the average distance of the agents. This is caused by the concave agents 'pulling' away from their neighbours. The baseline experiment shows a limited change in the variance (jitter) due to the swarm being close to stable even though there is a void present.

The concave reduction algorithm creates a more pronounced variation due to the void having multiple anomalies. The concave reduction process closes the void over a period of 3 seconds and the swarm then settles to a steady average distance with a stable variance. The increased variance up to 3 seconds is the 'percolation' of the internal anomalies to the outer perimeter as the void is closed.

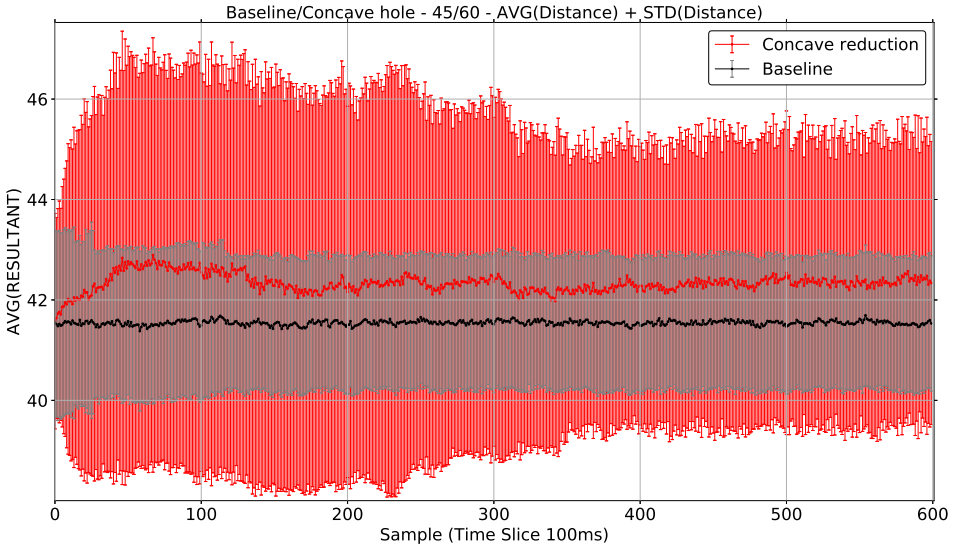


Fig. 6.22: Concave reduction stability effect distance

The residual jitter from 3.5 seconds on-wards (Figure 6.23) is the algorithm's effect on the outer perimeter of the swarm. The baseline shows a steadier average with a reduced variance as the basic swarming algorithm allows the agents to settle to a formation that is more structurally stable.

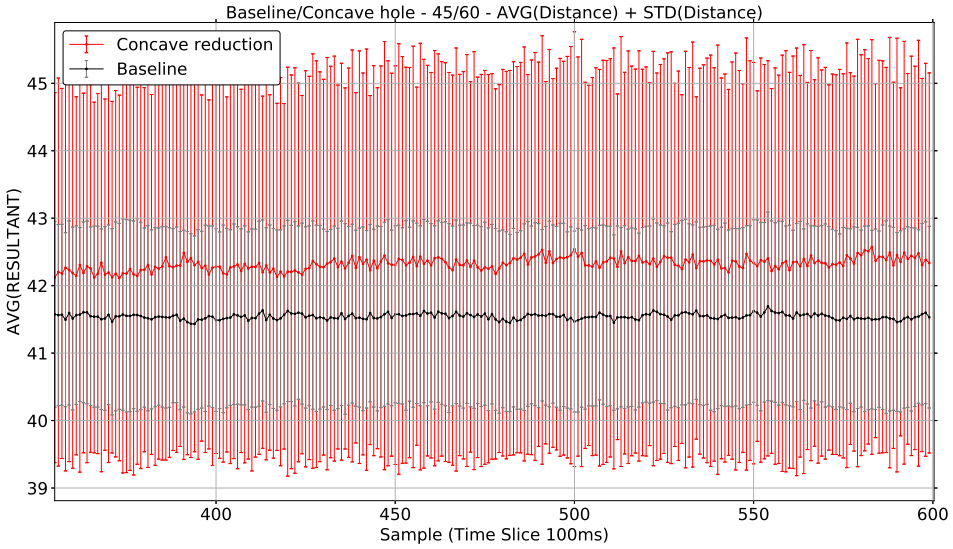


Fig. 6.23: Concave reduction stability effect distance

Figure 6.24 shows that although the algorithm has introduced jitter and therefore increased both the average *inter-agent magnitude* and the variance the resultant changes have not caused the swarm to become cohesively unstable. The magnitude and the variance never take the magnitude below 0. From 3.5 seconds the *inter-agent magnitude* fluctuates slightly with an increased variance caused by the re-positioning of the 'concave' agents. Although the agents are less structured the increased resultant magnitude ensures the swarm remains cohesive.

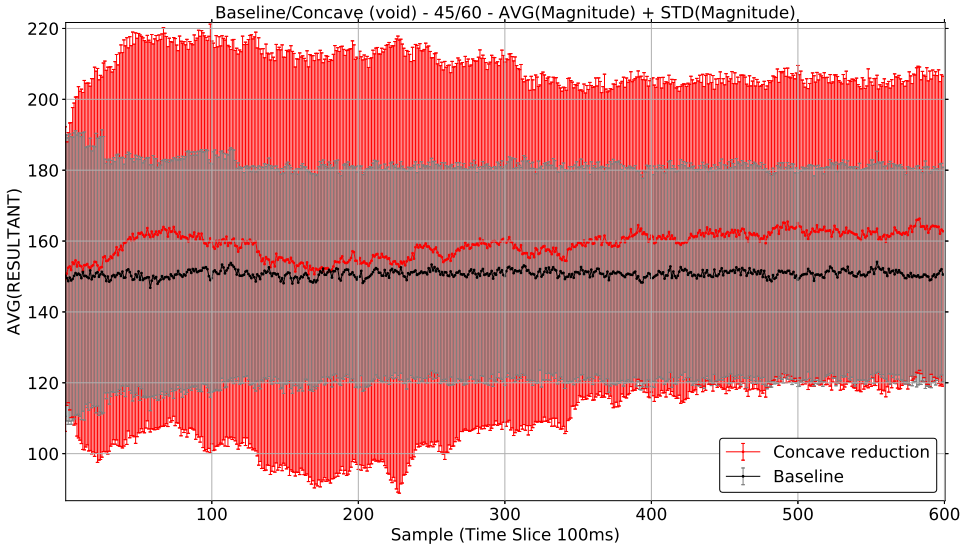


Fig. 6.24: Concave reduction stability effect magnitude

Once the void is removed the swarm settles to a more stable phase 3.5 seconds onwards (Figure 6.25), there is a slightly higher variance than the baseline which is caused by the concave reduction affected agents 'pulling' the swarm. This 'pulling' causes the agents to be slightly more distributed and therefore increases the inter-agent cohesion. There is also the addition of the 'snapping' effect which increases the variance.

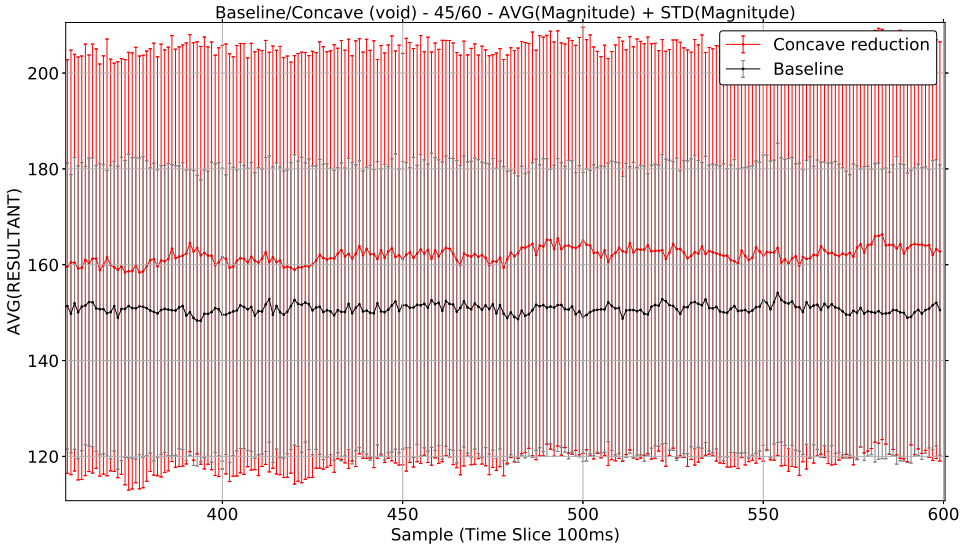


Fig. 6.25: Concave reduction stability effect magnitude

The effect on the swarm of removing the void is to reduce the overall number of perimeter agents. Figure 6.26 shows that as the void is removed from the swarm the number of perimeter agents falls. The change in size is caused by two processes. The swarm structure is being altered on the outer perimeter as the agents move towards a more circular formation and the internal agents identified as perimeter agents of a void move to reduce the internal anomaly.

Once the initial disorganised phase settles, which takes approximately 1.2 seconds, the effect of the concave reduction starts to take effect. The perimeter size starts to reduce. The majority of the reduction is the void shrinking. The swarm then goes through a settling period where the overall perimeter size of the swarm stabilises and eventually the residual snapping effect is left at the outer perimeter. This occurs at approximately 3.5 seconds into the simulation.

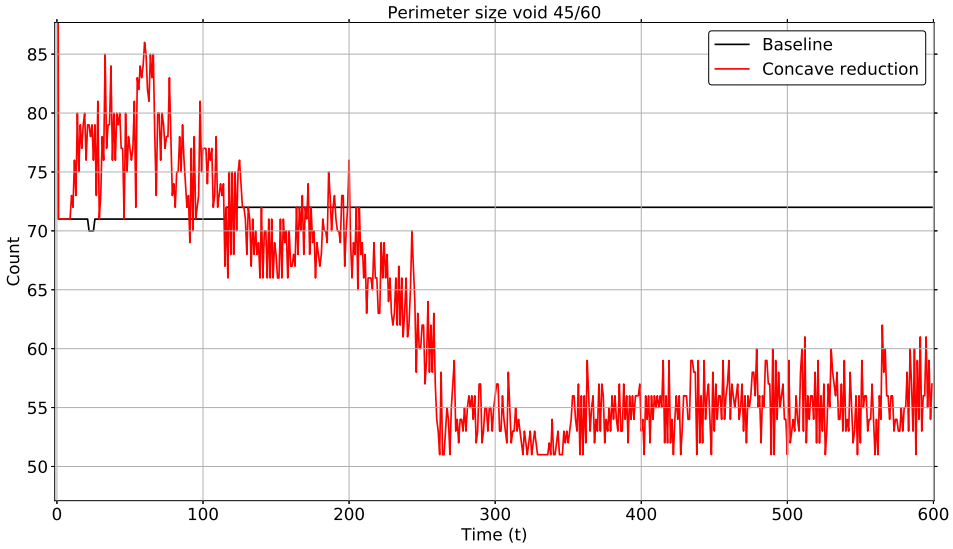


Fig. 6.26: Concave reduction perimeter effect

The effect on the swarm's structure is shown in Figures 6.27 and 6.28. Figure 6.27 shows the agents positions during the simulation for the baseline and the concave reduction enabled swarms. The baseline swarm is shown in black. The red traces are for the swarm using concave reduction. Figure 6.28 is a more detailed view highlighting the baseline lattice structure.

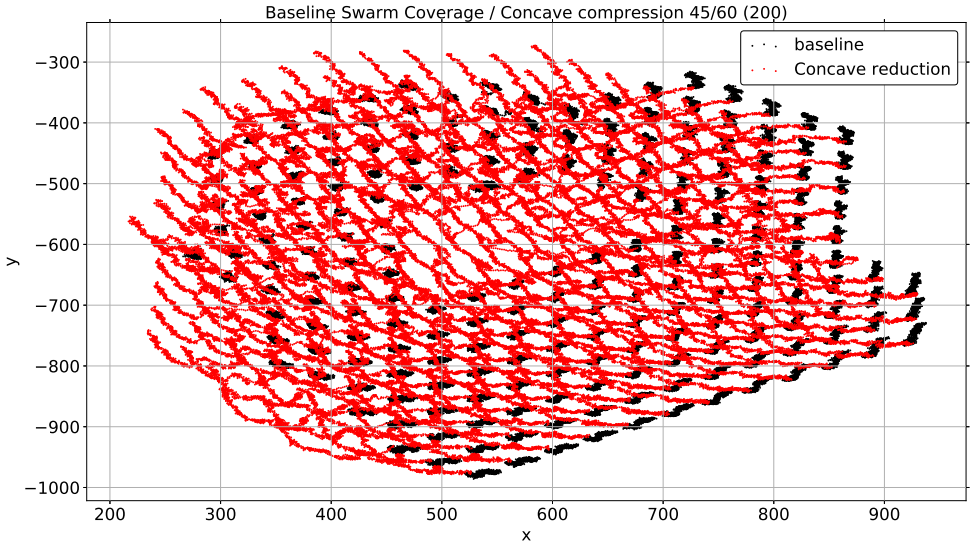


Fig. 6.27: Agent movement comparison

The overall effect of the concave reduction on the swarm's movement can be seen in figure 6.27. The void is closed by the reduction and the overall area of the swarm is reduced. There is however a negative effect with respect to the concave reduction; the swarm has a directional bias due to the large straight edge which causes the swarm to 'drift'. When looking closely at the positions (Figure 6.28) the baseline agents remain relatively static in their positions vibrating slightly to maintain the equilibrium of the internal magnitudes.

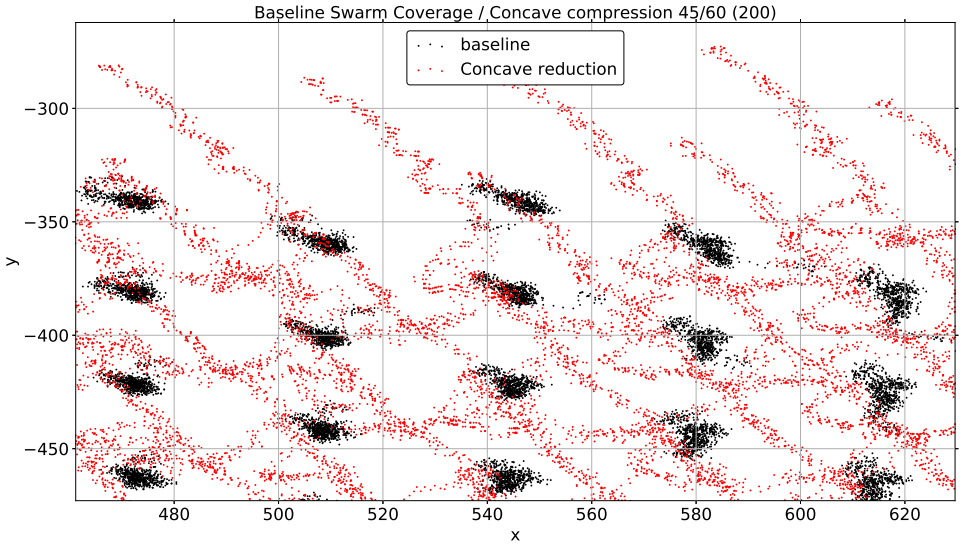


Fig. 6.28: Agent movement comparison

6.6 Concave reduction for object surrounding

Concave reduction has the benefit of creating a convex-perimeter and removing a concave perimeter. This effect can be applied to surrounding objects. As a void is removed agents still avoid obstacles, this results in a perimeter edge that tightly encloses an object's perimeter. The concept of detecting an object and surrounding it is not new. In 2013 Zhang et al [165] investigated this as a mechanism to assist in oil spillage containment. The process they used was based on ant-colony foraging. The agents initially carried out reconnaissance to detect a spillage, once a target is identified the agents use a communications infrastructure to inform nearby agents of the location of the spillage and the agents then use a *destination vector* to locate the spill. To surround the spill the agents move in an anti-clockwise manner following the perimeter wall of the target. This thesis uses a different approach to solve this same problem.

One problem with the above approach is that the swarm may not find the spillage due to the paths the agents take when foraging not intersecting with the spillage. Another problem is that the system does not consider multiple targets. Finally there is the issue of the swarm requiring a communications infrastructure.

This thesis focuses on arbitrary sized, low cost, swarms; this is a similar approach to the US Navy in the LOCUST project [156, 141]. Also the approach of using concave reduction removes the need for a communications infrastructure.

Consider an oil slick in an environment. This could be a section of open-water or a lake/reservoir. The solution is to deploy a swarm at the perimeter of the known area by a boat or at a shoreline if the area is small enough (Figure 6.29). The deployed agents, using local sensing and the concave reduction algorithm, are enabled. The swarm initially expands to an optimum distribution for the specified field effects. The *concavity reduction vector* will then reduce the deliberately created void and the swarm encapsulates the oil spill. If there are multiple spills within the area the void reduction process will still encapsulate the area because the algorithm is not dependent on communications and operates purely by logic.

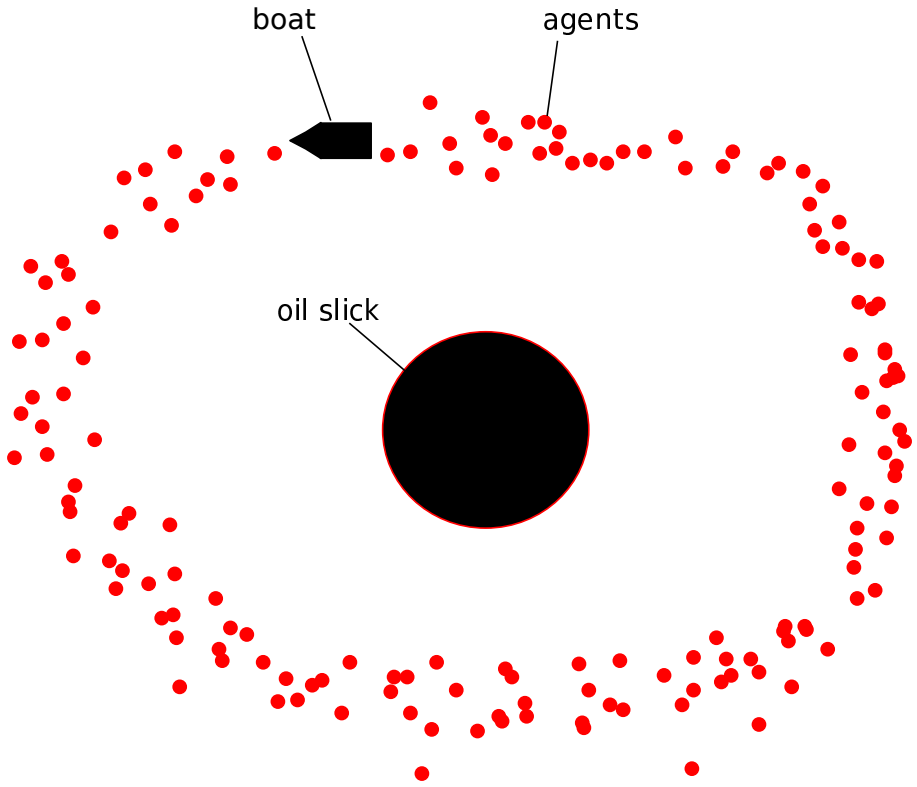


Fig. 6.29: Concave reduction oil slick surrounding

Figure 6.30 is a screen shot of the deployment within the simulator for testing this hypothesis.

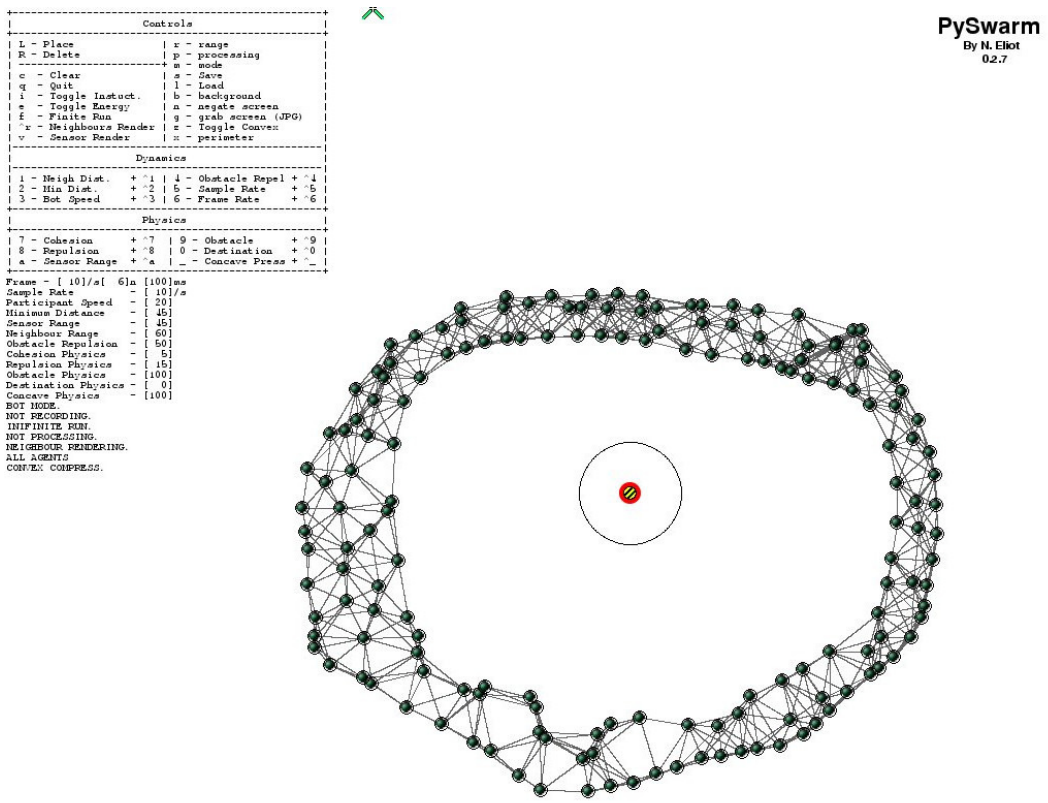
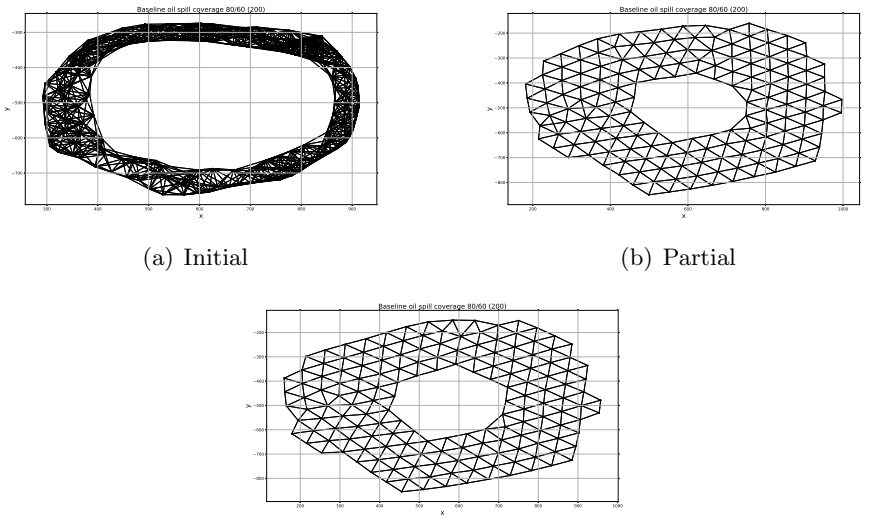


Fig. 6.30: Oil spill containment simulation

Figure 6.31 shows the containment process using the baseline configuration without concave reduction. The swarm expands due to the field effects and then stabilises into a swarm which contains a void area. The swarm stabilises and the swarm moves slightly as the cohesion and repulsion fluctuate to maintain the swarm's structure. However the void does not close and the containment process does not occur. The agents that do come in contact with the obstacle are repelled by the obstacle repulsion field.



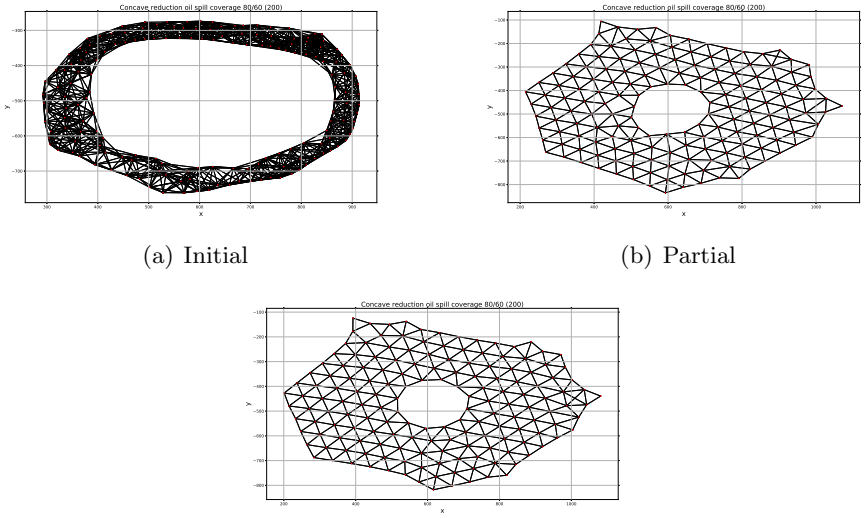
(a) Initial

(b) Partial

(c) End

Fig. 6.31: Baseline oil spill containment

Figure 6.21 shows the same simulation with void-reduction activated. The swarm expands as expected due to the field effects but in addition the concave reduction reduces the void within the swarm. This reduction in the void causes the swarm to shrink around the obstacle completely enclosing it at the obstacle perimeter. On the left hand side of figure 6.32(b) the percolation effect can be seen as described in § 6.5.



(a) Initial

(b) Partial

(c) End

Fig. 6.32: Concave reduction spill containment

Figures 6.33 and 6.34 show the effect of the concave reduction on the swarm's agent distribution compared to the baseline for the oil spillage scenario.

Figure 6.33 shows distance distribution of the swarm for both the baseline (grey/black) and the concave reduction (red). The baseline expands then settles at approximately 6 seconds. This is also the case for the the concave reduction swarm. The baseline swarm, following the initial expansion, remains relatively slow changing with respect to distance and magnitude. The concave reduction swarm is affected more significantly, at approximately 10 seconds into the simulation the swarm's internal void perimeter makes contact with the oil spillage (obstacle). This has the effect of disrupting the average distance and average *inter-agent magnitudes*. This effect diminishes slightly at approximately 18 seconds where the swarm's *concavity reduction vectors* cause the swarm to surround the spillage. The surrounding is followed by a few remaining changes caused by the agents at the spillage perimeter 'snapping', the surrounding process is complete.

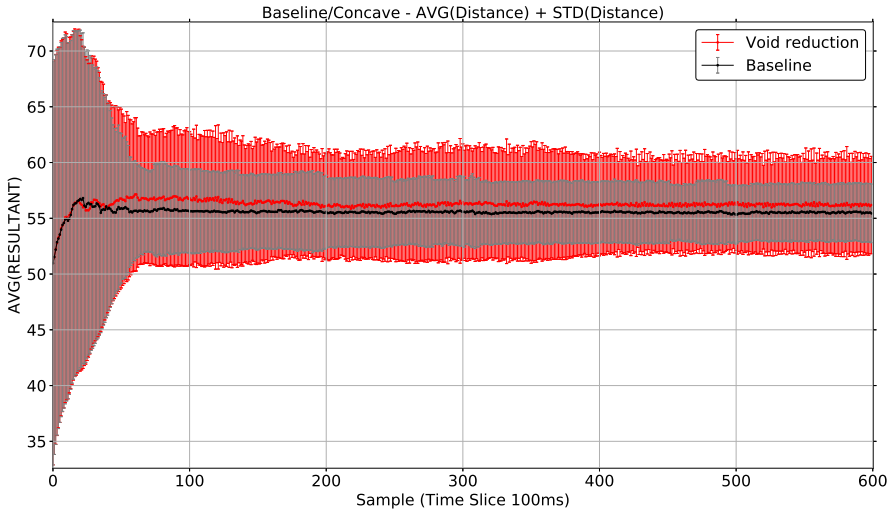


Fig. 6.33: Oil spill containment distance

Figure 6.34 shows the comparison of the *inter-agent magnitudes* for the baseline and concave reduction swarms. The initial deployment of the swarm is so condensed that the average *inter-agent magnitude* is negative which indicates a high level of expansion. Within 2 seconds the expansion has reached a point where the average magnitude is positive which indicates the swarm is cohesive and will therefore remain as a single entity and be capable of surrounding an object without breaking apart.

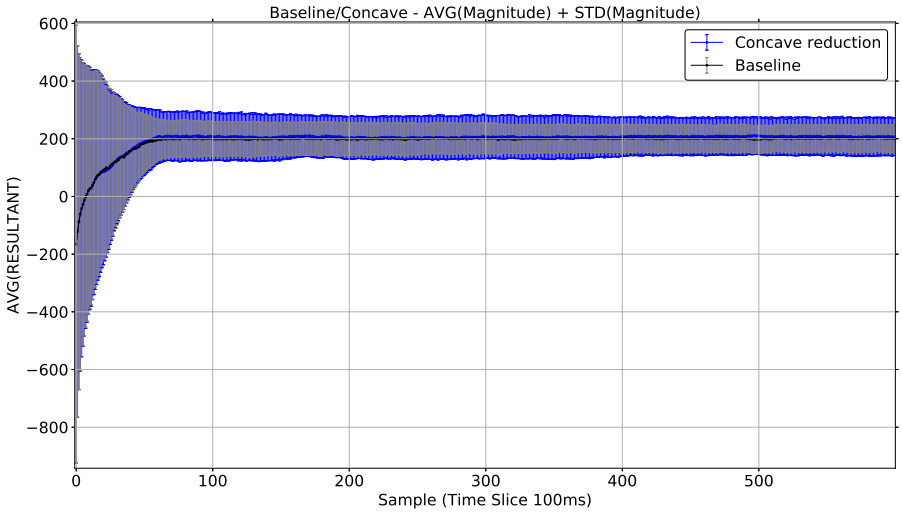


Fig. 6.34: Oil spill containment magnitude

When the swarm 'shrinks' to surround the obstacle there is an erratic change in the number of perimeter agents. Figure 6.35 shows the number of perimeter agents identified over the duration of the simulation. The graph shows the baseline swarm perimeter size decreases steadily and then settles (The swarm has not enclosed the spillage). The perimeter count has settled but as shown in figures 6.33 and 6.34 the agents are still moving (magnitude variance and magnitude >0) but the movement does not effect the overall structure.

In the case of the concave reduction swarm the perimeter size is erratic due to the snapping effect (Figure 6.11). When the swarm encounters the obstacle at approximately 10 seconds there is a change due to 'snapping' as the agents 'fold' around the obstacle. The perimeter size then continues to fall gradually as the void is percolated out of the system, the perimeter size then stabilises as the obstacle is fully surrounded.

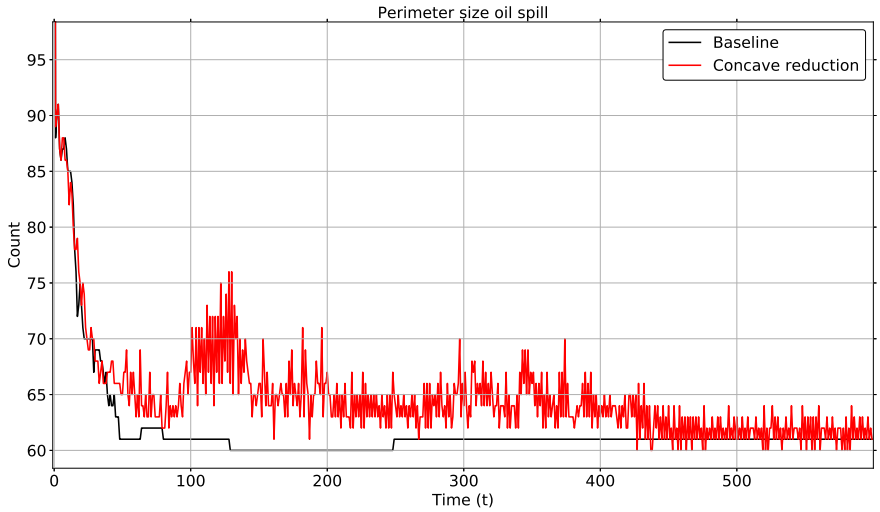
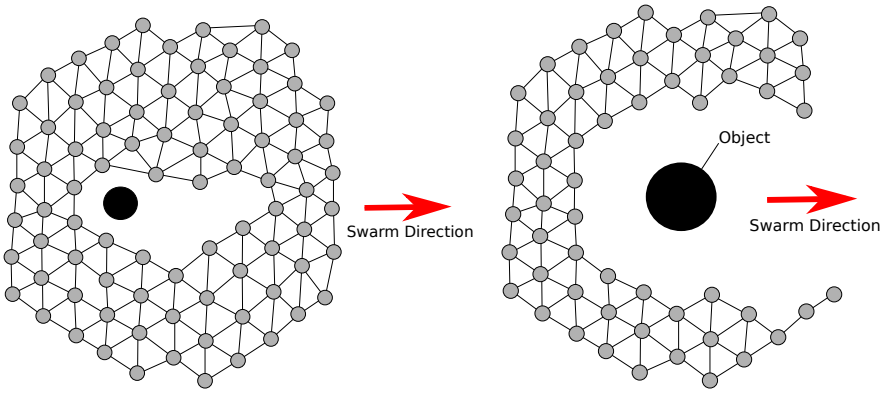


Fig. 6.35: Swarm perimeter size comparison

6.7 Concave reduction for destination-based swarms

A goal based swarm that is travelling between two points can have its path disrupted by an obstacle. This event can cause a swarm to develop an anomaly in the form of a void (Figure 6.36(a)) due to agents temporarily separating as the swarm passes the obstacle or the swarm may split entirely if the obstacle is very large (Figure 6.36(b)).

When the object is small the cohesion and repulsion fields may make the swarm naturally reform (Figure 6.36(a)), however as the swarm progresses past the obstacle a void is created at the forward edge of the obstacle. If the purpose of the swarm is to deploy fertilizer over crops or carry out a reconnaissance task then the required full area coverage is not achieved. This failure to achieve total coverage reduces the effectiveness of employing a swarm to carry out tasks of this type.



(a) Swarm distortion via an object creating a void

(b) Swarm distortion via a large object

Fig. 6.36: Swarm perimeters and voids

Concave reduction has been shown to remove voids in static swarms in § 6.5. Concave reduction can also be applied to a goal-based swarm to control coverage when passing small object: 'small' means the maximum radius of the obstacle is of the order of the size of the neighbour field of an agent or less. When concave reduction is applied to the swarm it will surround the obstacle. Surrounding the obstacle creates a greater degree of coverage and increases the effectiveness of the swarm.

Taking the baseline swarm from chapter 5 the swarm environment can be modified to include an obstacle in the path of the swarm. Table 6.5 shows the weightings and field effects for the baseline and concave reduction parameters for the experiment to demonstrate the concave reduction coverage improvement. Figure 6.37 shows the swarm and object setup within the simulator for the experiment.

Weight component	**Baseline swarm**	**Description**
Sample rate	100	ms - Unit sampling interval
k_{cr}	100	weight adjuster for concavity vector
k_o	100	weight adjuster for obstacle field
k_c	5	weight adjuster for cohesion field
k_r	15	weight adjuster for repulsion field
k_d	35	weight adjuster for destination vector
Repulsion Boundary	45	units
Neighbour Distance	60	units
Obstacle size	60	units
Speed	20	units/s

Tab. 6.5: Swarm coverage parameters

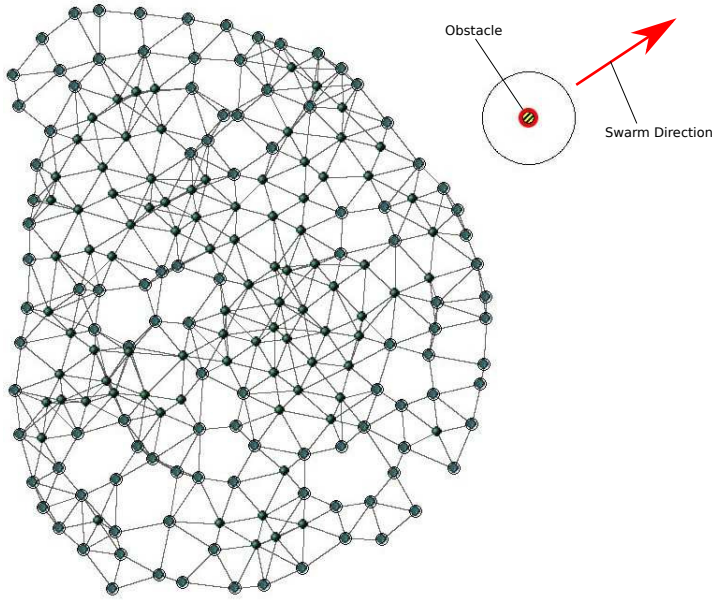


Fig. 6.37: Initial configuration

The experiment was run for 60 seconds to determine how the swarm is affected by the obstacle. The experiment is executed twice once with and once without concave reduction enabled. For figures 6.38 and 6.39 the blue lines are generated by plotting the position of each agent as they progress towards the goal. Figure 6.38 is a path trace of the swarm as it propagates around the obstacle without concave reduction. The agents are repelled by the obstacle and the leading edge 'flows' around the perimeter. Due to the compression caused by the agents passing around the obstacles an expansion is experienced at the trailing side and the agents are 'pushed' such that the swarm reforms. However the time it takes for the compression to be removed causes the swarm to create a void at the forward side of the obstacle.

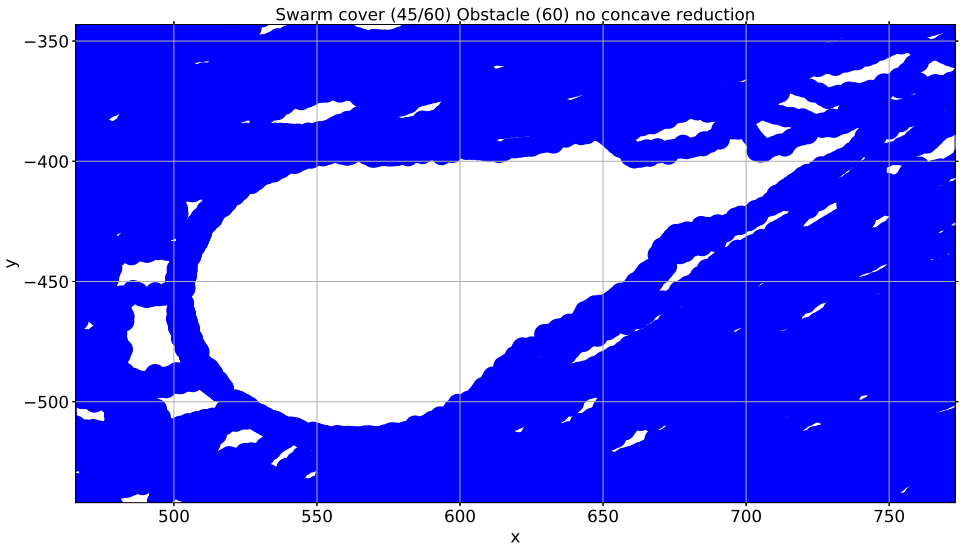


Fig. 6.38: Swarm without concave reduction (repulsion field 60 units for obstacle)

Figure 6.39 shows the path trace for the same environment settings with concave reduction enabled. In a similar way to the baseline experiment the leading edge agents approach the obstacle and through the repulsion effect of the obstacle the agents travel around the outer edge. Once the agents have passed the mid point of the obstacle the swarm splits in a similar way to the original experiment. The void is created in a similar way but when the split segments converge a change occurs. The meeting edges create a concave edge and the *concavity reduction vectors* affect the agents by 'pulling' the 'edges' together. This effect propagates back along the edges until the concave perimeter is reduced back to the obstacles forward edge. This effect slows the progress of the swarm's internal movement but closes the void. As the swarm progresses towards the destination the back portion of the swarm moves around the obstacle and swarm continues onto the destination.

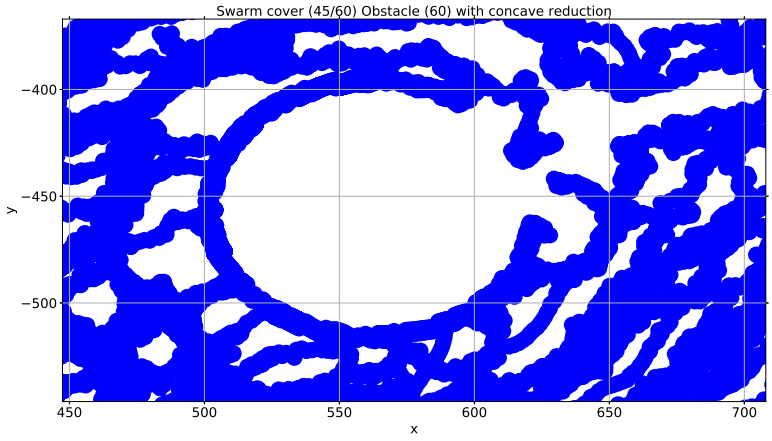


Fig. 6.39: Swarm with concave reduction (repulsion field 60 units for obstacle)

6.8 Conclusion

This chapter has demonstrated the concept of controlling a swarm's structure through the identification of perimeter anomalies in the form of concave edges. The identification of the anomalies is achieved locally by an individual agent without needing any communications infrastructure. The technique works with arbitrary sized swarms demonstrated here experimentally with a swarm of 200 agents.

Concave reduction is shown to work with static swarms such that a swarm can stabilise to a more regular shape. It is possible for a swarm to develop where no concave reduction will be in operation which would allow the swarm to refer back to a baseline condition. Figure 6.40 shows an ideal swarm shape that would result in no concave reduction based movement.

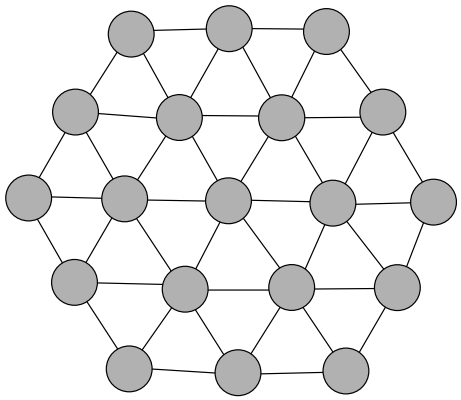


Fig. 6.40: Swarm structure with no concave reduction

The chapter demonstrates by experiment that by altering the vector calculations for an agent based upon a set of stimuli it is possible to improve the coverage of an area and also create a 'compression' effect that can increase the application landscape for swarming technologies that incorporate concave reduction.

7. SWARM COORDINATION - AREA FLOODING

Area flooding is a technique used to fill an enclosed area with agents such that they are distributed as 'effectively' as possible throughout an area. An optimum distribution of agents is not necessarily an even distribution. The distribution of the agents is governed by the field effects and the interaction of agents with obstacles (*interaction vectors*).

Filling an area can be applied to tasks that require an unknown environment to be analysed or surveyed. Consider a disaster area following a landslide or a building collapsing following an earthquake; the movement in the land and buildings will produce spaces that are unmapped. The unmapped areas may require investigation to locate people, resources or to create some form of sensor network to analyse the conditions within the area such as creating a heat map or to identify the location toxic gases.

This thesis demonstrates two swarm expanding techniques using field effects to perform area filling for the purposes described above.

The concept of using a swarm to provide coverage over unknown areas is a current area of research. Alvissalim et al. (2012) discuss the application of commercially available drones to provide a communication infrastructure across an unknown disaster areas [2]. Scheutz and Bauer (2006) use both cohesion and repulsion as a mechanism to create coverage of an area that requires protection in an adversarial environment [132]. The technique used by Scheutz and Bauer (2006) is similar to the principle of *goal-based swarms* as discussed in chapter 5. In their paper they do not use cohesion to ensure the swarm remains a cohesive unit, rather they have all the agents using a *destination vector* to cause agents to converge on a target and the repulsion vector prevents collisions. The cohesive effect is not required due to all agents having a common goal with no requirement for the agents to remain in close proximity during the terrain traversal. Ramaithitima et al. in 2015 and 2016, working as part of Vijay Kumar's research group at the University of Pennsylvania, use the idea of filling an area through a deployment

strategy that requires the swarm to identify the placement of agents in a given space. The agents are then 'shuffled' into an identified position. The technique applied in 2015 [121] is based upon coverage path planning and involves adding agents when required and does not require a global positioning reference. In their 2016 paper [122] they use Voronoi graphs to achieve the same effect. Yang et al. (2015) [159] also use Voronoi graphs to achieve their swarm coverage goal. The use of Voronoi graphs requires inter-agent communications. Schroeder and M. Kumar (2016) use the bio-inspired swarming technique of creating chemical trails (pheromones) combined with a foraging approach which they refer to as a 'food foraging' technique [134].

The techniques used in this thesis are similar to Schroeder and M. Kumar in that the agents do not require a global positioning system and differ from Alvissalim et al., Scheutz and Bauer, and Ramaithitima et al. in that the space filling is achieved with a fixed sized swarm.

The expansion fill is implemented using two techniques. The main principle behind both techniques is to increase the repulsion field effect of the agents over time to increase the area coverage of the agents. The expansion fill can use both the cohesion and repulsion field effects in a similar manner to a static swarm in free space. This technique ensures all the agents remain in close proximity and as far as possible the agents form a single entity. Alternatively the expansion fill can be implemented using only a repulsion field effect. Cohesion can be eliminated in an enclosed space as the agents have a limited range of movement (bounded by a perimeter). When agents move to a repulsion boundary, either an area perimeter or an obstacle, they are repelled. As the space is finite the repulsion 'pushes' the agents back into the swarm countering the expansion that is induced by the *interaction vectors*. If the repulsion field effect creates a swarm that reaches all the boundaries the repulsion imposes a compression effect as the *interaction-vector* 'pushes' the agents into the boundaries.

7.1 Field effect modification with cohesion and repulsion

The concept behind the field effect expansion is to increase both the repulsion and cohesion fields over a period of time. This increase in the field effects makes the agents increase the distance between each other expanding the swarm as a whole. The expansion increases until, due to boundary compression, the swarm is unable to expand further. In an extreme case the expansion will result in a set of field effects that create a mesh based

swarm structure rather than the desired hexagonal structure that the field effects should create. Identifying either the change in modality or the inability to expand, which can be identified by the *inter-agent vector magnitude* increase or a reduction in the inter-agent distance stability indicates area saturation has occurred.

To test this hypothesis a swarm is modelled in the simulator. The model consists of an obstacle-based enclosed space and a swarm consisting of 60 agents. The experiments parameters for the simulation are shown in table 7.1

Parameter	**Value**	**Description**
k_c	5	weight adjuster for cohesion bias
k_r	60	weight adjuster for repulsion bias
Sample rate	100ms	proximity sensor rate
Speed	20 units/s	agent speed

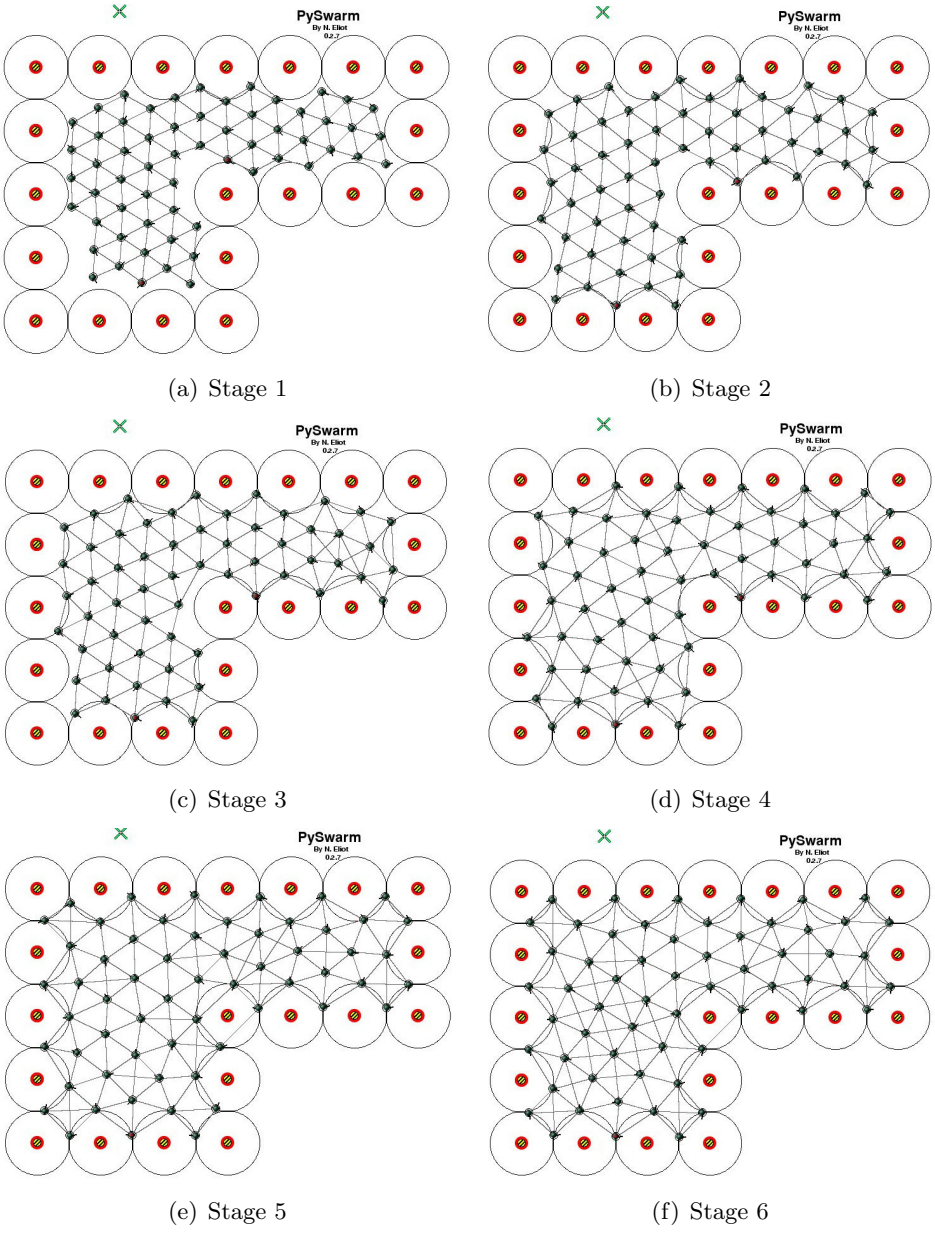
Tab. 7.1: Swarm parameters

The field effects are incremented in turn, neighbour range followed by repulsion range. After each repulsion field change the swarm is allowed to redistribute itself and stabilise. Table 7.4 shows the field effect settings that are used for the simulation. The field effect are selected so as to ensure the swarm parameters have the potential to create a hexagonal swarm.

Weight component	**1**	**2**	**3**	**4**	**5**	**6**	**7**	
Repulsion Boundary	50	-	60	-	70	-	80	units
Neighbour Distance	60	70	-	80	-	90	-	units

Tab. 7.2: field effect expansion sequence

Figure 7.1 shows the stages that the swarm progresses through during the simulation. Figure 7.1(a) shows the initial deployment of the 60 agents within the enclosed environment. Figure 7.1(f) shows the stage at which the experiment terminates due to area saturation.



(a) Stage 1 (b) Stage 2

(c) Stage 3 (d) Stage 4

(e) Stage 5 (f) Stage 6

Fig. 7.1: Space filling via field effect expansion

After the initial deployment the system is allowed to settle (Figure 7.1(b)) following the settling period the parameters are increased and the swarm settles again. Figure 7.1(c) shows how the change in parameters can cause the system to become unstable, the top right section of the swarm has became multi-modal yet there is still a section of the swarm that has not expanded to the boundary perimeter. As the swarm stabilises

this compressed section of the swarm 'pushes' through the swarm and the swarm increases in volume filling more of the space (Figure 7.1(d)). This process continues until changes in the parameters are unable to increase the distribution of the agents and the distances remain almost constant. Increasing the field effects further causes the swarm to become multi-modal as shown in 7.1(e) and 7.1(f).

These effects can be identified through a combination of the *inter-agent magnitude metric* and the distance metric.

7.1.1 Magnitude analysis

Figure 7.2 shows the resultant magnitude for the swarm for the entire simulation. Between 100 seconds and 105 seconds there is a significant change in the magnitude where the value becomes negative. This indicates there is a compression effect on the swarm and it is unable to expand the inter-agent distances.

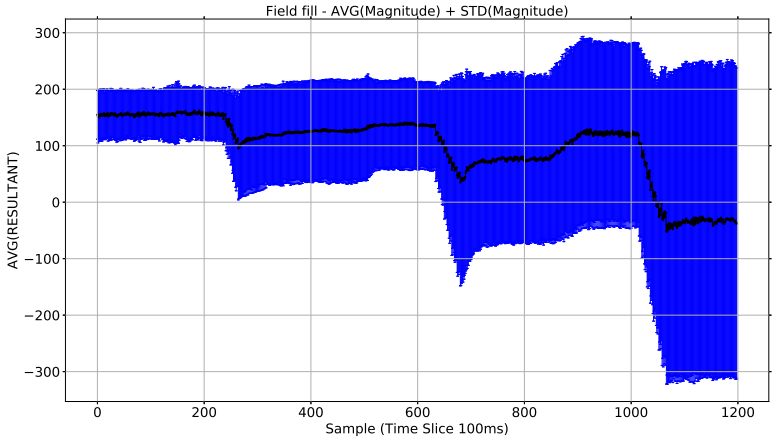


Fig. 7.2: Magnitude metric 0-120 seconds

Figure 7.3 shows the initial deployment status of the swarm and the first repulsion increase at 25 seconds. When the repulsion field is increased, the average *inter-agent magnitude* falls and the variation increases. The agents redistribute themselves which takes until approximately 65 seconds and a new baseline is established for the field effects.

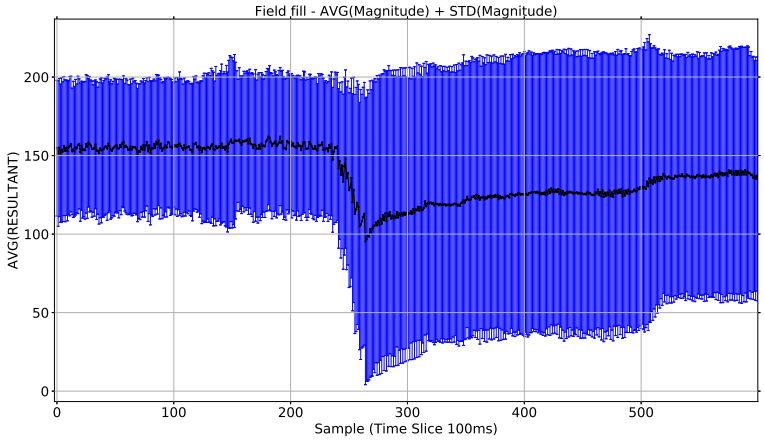


Fig. 7.3: Magnitude metric 0-60 seconds

The next increment at 65 seconds creates a similar reaction to the magnitude and variance and the swarm settles to another baseline, however, the new baseline has a variance which drops the resultant magnitude to below zero. This indicates that some of the swarm is experiencing a compression effect which will result in the swarm body moving towards an uncompressed area. At 85 seconds the next repulsion increase is made and the reaction of the swarm metrics indicates something has changed in how the swarm is reacting. The average *inter-agent magnitude* rises and the variance increases. This change is the effect of an increase in the modality of the swarm which indicates that the swarm is fully distributed within the environment as the neighbour field effect is detecting additional neighbour agents. The final increment at 108 seconds creates an even greater increase in the variance, this is caused by a further increase in the modal distribution of the agents and the swarm is being compressed by the boundary of the space.

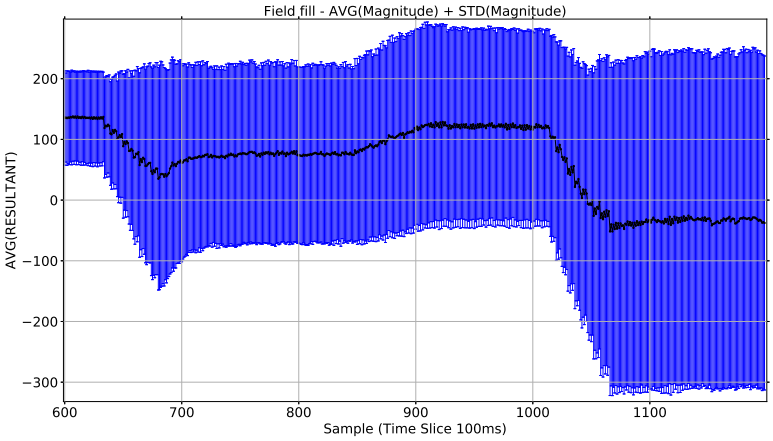


Fig. 7.4: Magnitude metric 60-120 seconds

7.1.2 Distance analysis

Figure 7.5 shows the distance metric for the simulation. The initial deployment is shown at 0 seconds followed by incremental changes in the distances and variance of the distances as each increment is made in the field effects. The simulation terminates after 12 seconds.

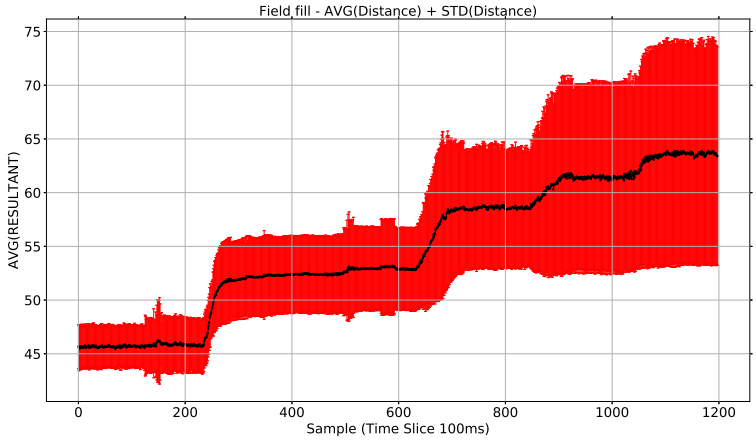


Fig. 7.5: Distance metric 0-120 seconds

Figure 7.6 shows the initial expansion increasing the field distance causing the agents to 'spread' throughout the space. Each increment having a settling period as the swarm expands.

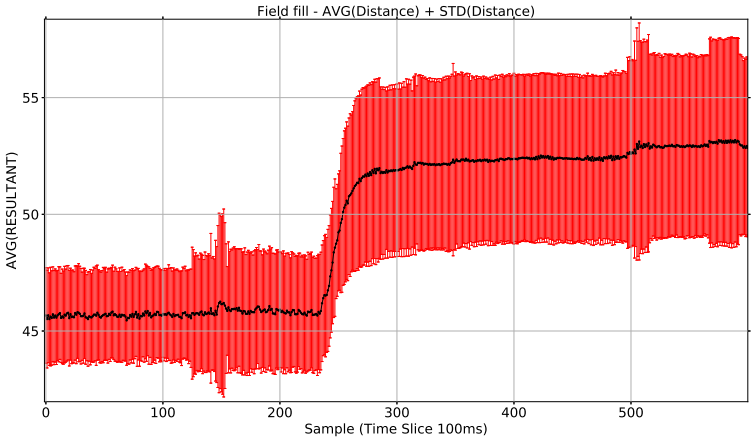


Fig. 7.6: Distance metric 0-60 seconds

Figure 7.7 shows the simulation from 60-120 seconds. Between 60 and 85 seconds an increment is made in the field effects and the swarm expands. At 85 seconds there is further increment, however the effect on the distribution is different. The distribution of the agents has changed due to the modality of the swarm. The average distance has increased and there is an increase in the variance. This would indicate that the swam has fully expanded and is unable to expand further. The final increment at 105 seconds can be seen to further impact on the modality indicating the area is saturated.

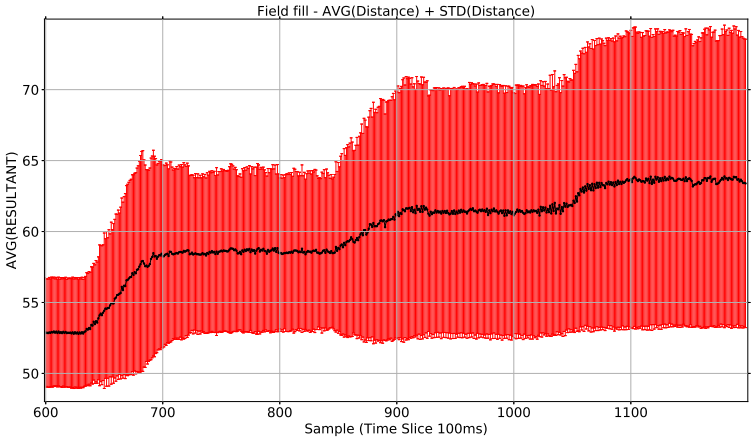


Fig. 7.7: Distance metric 60-120 seconds

7.1.3 Combined magnitude and distance analysis

Combining the results of the distance and *inter-agent magnitude* metrics provides a more in depth view of the effects changing the field effects has upon the swarm's structure.

There is a significant change in the *inter-agent magnitude* when the field effects are incremented to 70 units for repulsion and 80 units for neighbour distance (Figures 7.9 and 7.11). There is no significant change in the inter-agent distances (Figures 7.8 and 7.10). Until this point the average *inter-agent magnitude* is positive which indicates an aggregate cohesion in the swarm. After the increment there is an aggregate repulsion in the swarm indicating the swarm is being compressed and the repulsion field effect vector magnitude is greater than the cohesion field effect vector magnitude. A positive cohesion does not indicate the space is not filled only that the swarm still shows a tendency to remain a cohesive entity.

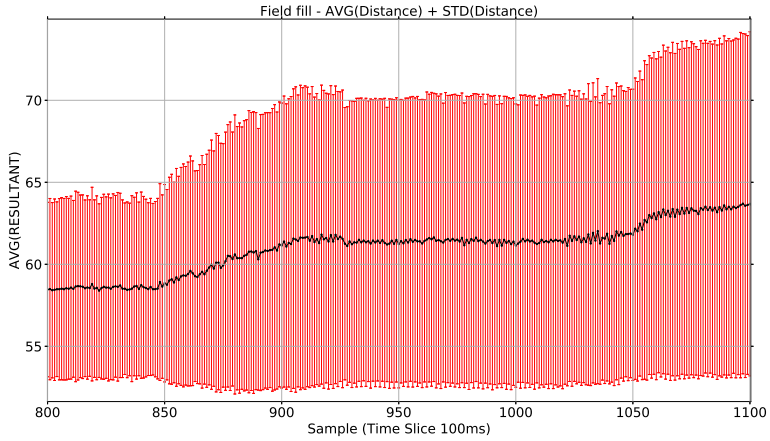


Fig. 7.8: Distance metric 80-110 seconds

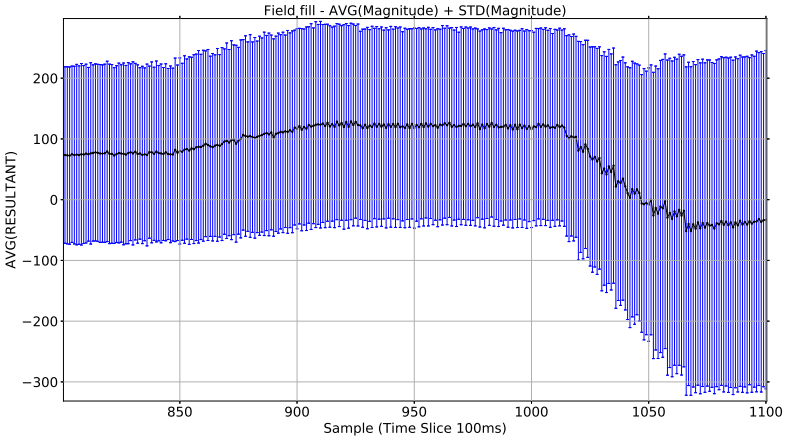


Fig. 7.9: Magnitude metric 80-110 seconds

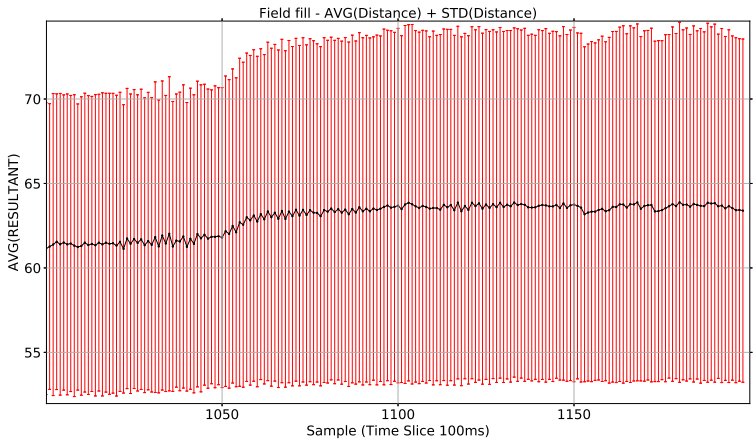


Fig. 7.10: Distance metric 100-120 seconds

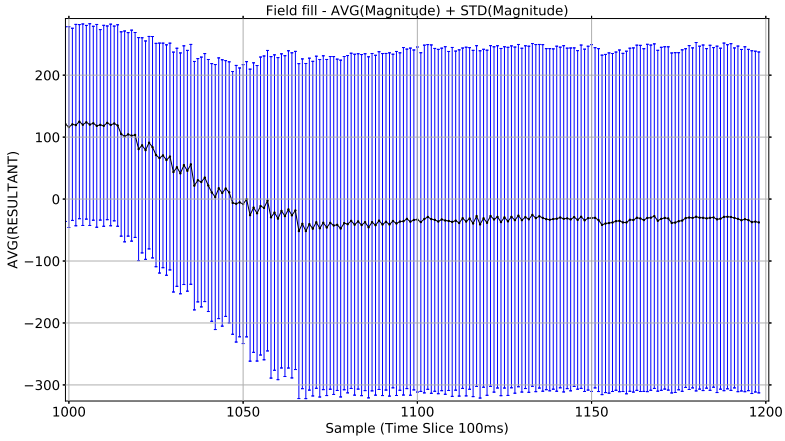


Fig. 7.11: Magnitude metric 100-120 seconds

7.2 Field effect modification with repulsion only

The previous mechanism for area filling used both repulsion and cohesion to ensure the swarm remains, as far as is possible, a single entity. When using repulsion only it is accepted that the agents are in a restricted area and not being cohesive is not and issue as the field effects can be expanded until the agents interact with each other and

the boundary. A similar approach was used by Scheutz and Bauer in 2006 [132]. The repulsion field effect in this thesis is used with a fixed sized swarm and the repulsion field is increase over time. The simulation for this experiment uses 52 agents in the same confined space as the previous experiment. The parameters are shown below:

Parameter	**Value**	**Description**
k_c	0	weight adjuster for cohesion vector
k_r	15	weight adjuster for repulsion vector
Sample rate	100 ms	proximity sensor rate
Speed	20 units/s	agent speed

Tab. 7.3: Swarm parameters repulsion only

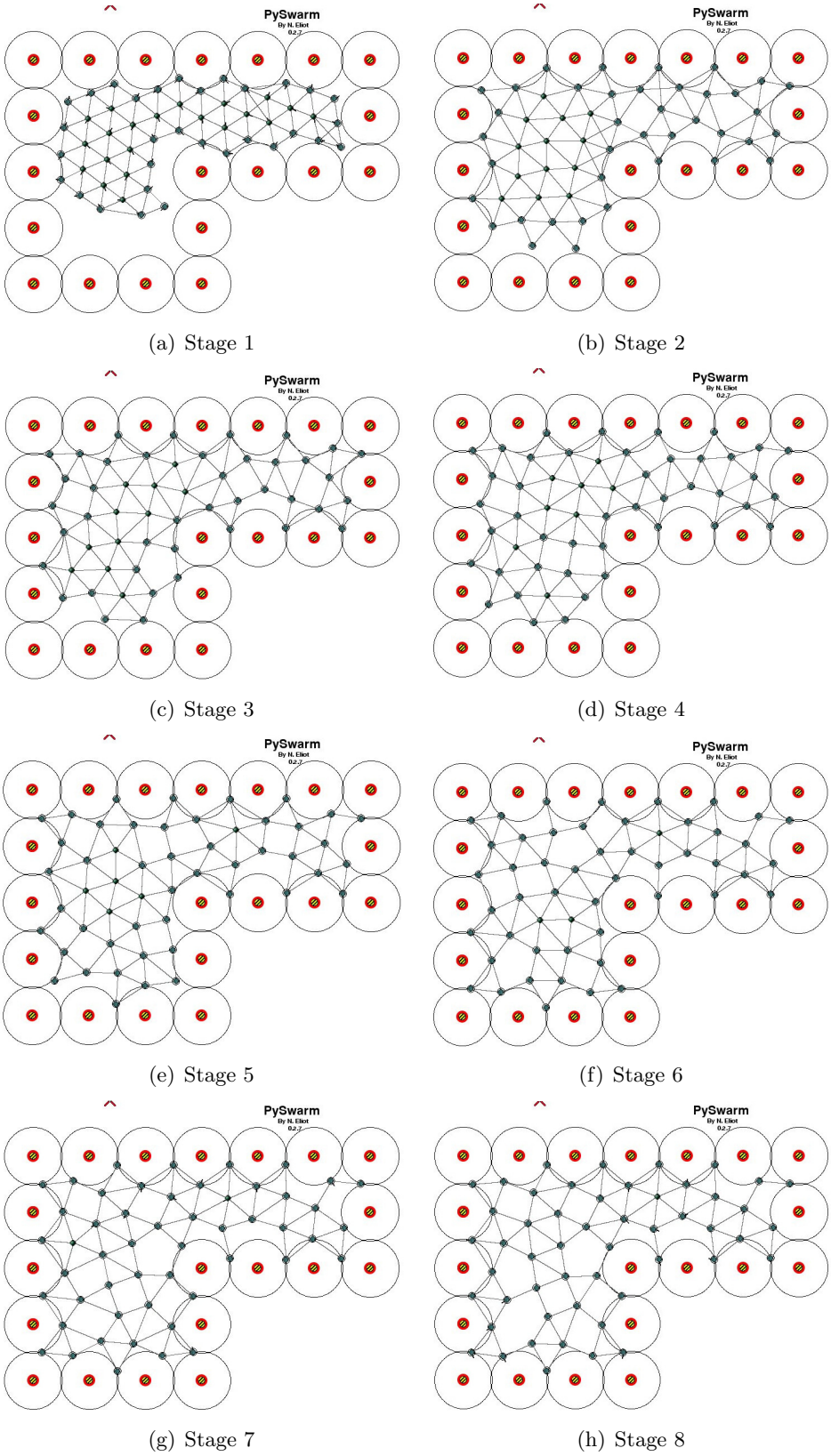
Weight component	**1**	**2**	**3**	**4**	**5**	**6**	
Repulsion Boundary	50	51	52	53	54	55	units
Neighbour Distance	70	70	70	70	70	70	units

Tab. 7.4: field effect expansion sequence

The theory behind this type of flood filling is as follows; Cohesion and repulsion acting against each other causes jitter, removing the cohesion allows the swarm to expand to its maximum volume with all agents distributed so they no longer interact at which point jitter will cease. The boundary created by the enclosed space will effect the expanding swarm by repelling the agents back into the swarm preventing expansion therefore the swarm movement will propagate to fill vacant areas. Jitter will initially be seen as the swarm equalises and it will decrease to zero when the swarm is fully expanded. There will be a point in the swarm's expansion when the agents will not be able to extend to a zero repulsion point and there will be a permanent interaction between the perimeter and agents. This condition is the exit point for the repulsion flood fill.

Figure 7.12 shows the stages that a field-repulsion-area-fill propagates through. The initial deployment is shown in Figure 7.12(a). When the simulation starts the swarm immediately expands (Figure 7.12(b)) and all the agents stabilise to a position where their movement stops. The field effect is increased and the swarm distribution movement

commences. This cycle continues until the swarm is unable to resolve to a neutral expansion point.



(a) Stage 1 (b) Stage 2

(c) Stage 3 (d) Stage 4

(e) Stage 5 (f) Stage 6

(g) Stage 7 (h) Stage 8

Fig. 7.12: Space filling via repulsion

7.2.1 *Inter-agent magnitude* analysis

Figure 7.13 shows the *inter-agent vector magnitude* produced between the agents. The initial deployment of the swarm shows a large negative value due to the swarm being in a highly compressed state. This causes the swarm to expand rapidly. Following the initial expansion the agents enter a phase of very close adjustment as their positions move towards a maximum expansion point. Eventually the agents are distributed to their maximum positions with the *inter-agent vector magnitude* stabilises to zero with no variation. At this stage the swarm has stopped moving. This effect cannot be detected by using the distance metric which will simply show a distribution distance with a fixed variance. Having a fixed variance does not indicate the agents have stopped moving as agents could be moving in sympathy to each other creating a net change of zero.

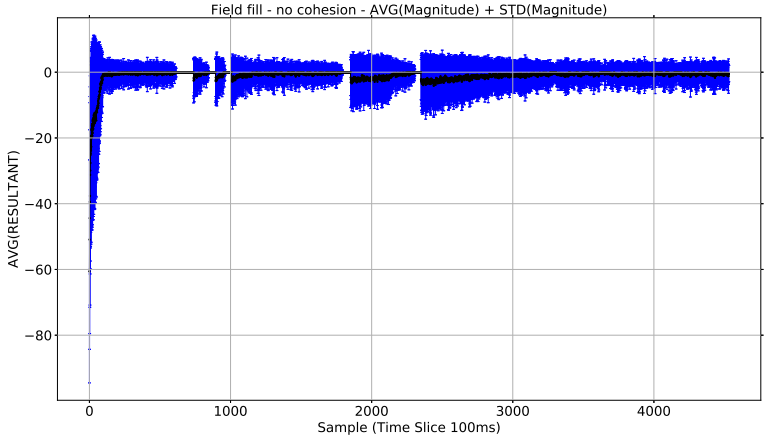


Fig. 7.13: Magnitude metric 0-450 seconds

Figure 7.14 shows the initial settling period of the swarm in greater detail between 0 and 65 seconds. The graph shows that the settling period lasts for approximately 55 seconds following the initial expansion of the swarm from 0 to 10 seconds. The initial deployment of swarm has a highly negative *interaction vector magnitude* causing rapid expansion. From 10 second until approx 62 seconds the agents *inter-agent vectors* cause the agents to spread within the space. When the agents reach the limits of the repulsion field effect they stop moving. The *inter-agent vector magnitude* stabilises to zero and all the agents stop moving.

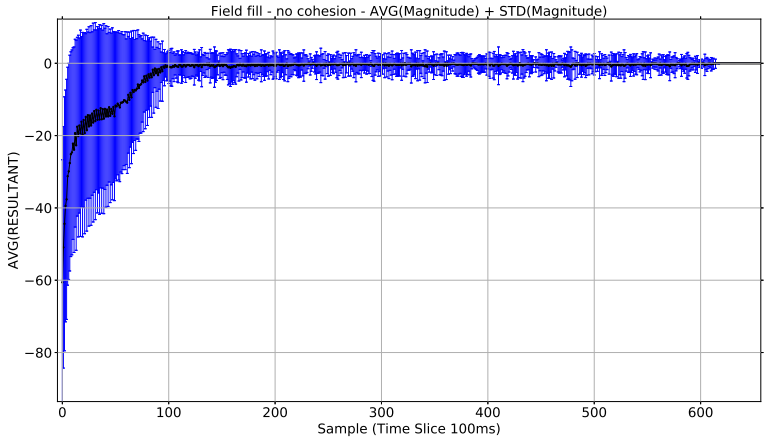


Fig. 7.14: Magnitude metric 0-65 seconds

Figure 7.15 shows a detailed view of the magnitude for the swarm at the first repulsion field increment at approximately 73.5 seconds. The repulsion field effect is incremented by 1 unit. The field effect change is sufficient to allow agents to be within the neighbour field effect range. The agents are already distributed from the initial expansion so there is a 'trickling' movement that occurs as the agents increase the area of the swarm within the space. The area containment is not causing any disturbance to the swarm at this stage so the swarm settles to zero (84.5 seconds). As with the initial expansion all the agents are distributed such that they are on or beyond the limits of the repulsion field effect and the swarm stops moving. This process of incrementing the field effect and the swarm stablising continues until the field effect causes the swarm to expand to a point that it fills the area and the agents are unable to move sufficiently to stop interacting.

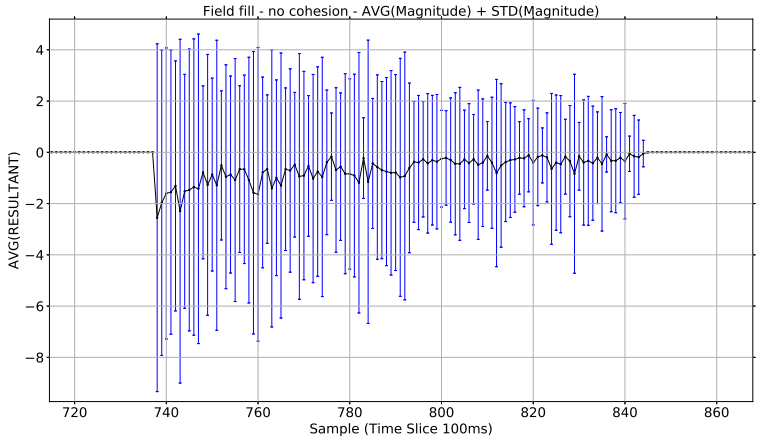


Fig. 7.15: Magnitude metric 70-85 seconds

Figure 7.16 shows the final part of the simulation where the swarm's resultant magnitude is unable to stabilise completely. The graph shows the increment at approximately 230 seconds creating swarm movement which slowly increasing towards zero as the swarm expands however the *inter-agent vector magnitude* is not able to reach zero. The swarm's agents are bound by the space and cannot distribute themselves to a point where the field effect overlap is overcome. The *inter-agent vector magnitude* average remains below 0 indicating there is a 'pressure' that cannot be overcome and the agents are in continuous movement. At this stage it is possible for all the agents to obtain a state of equilibrium and for the average magnitude to stablise but owing to their containment the average magnitude would not be able to rise to zero. The swarm has therefore expanded to fill the space.

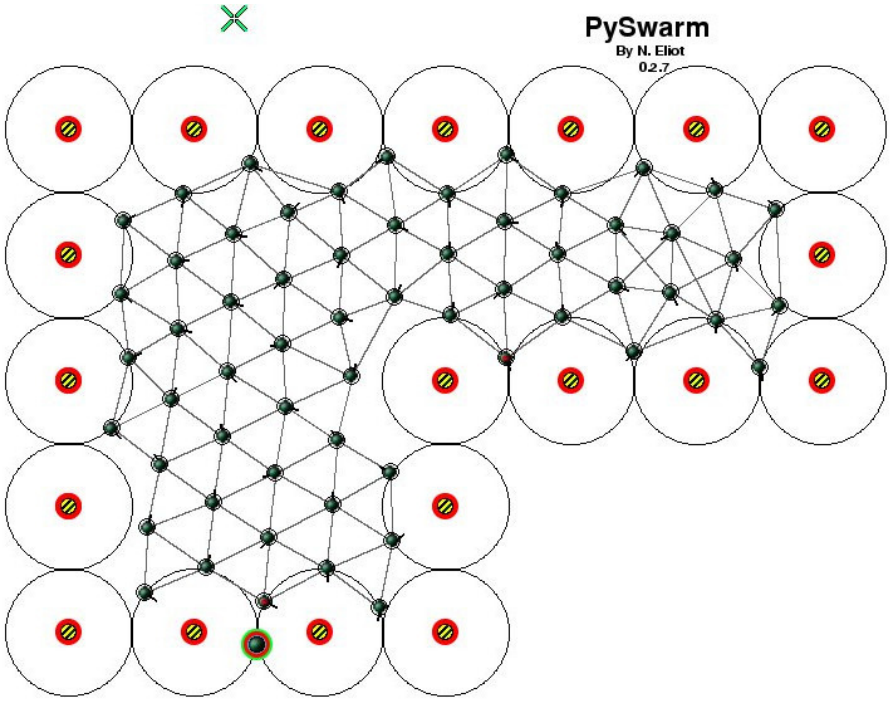


Fig. 7.16: Magnitude metric 235-315 seconds

7.2.2 Distance analysis

Figure 7.17 shows the distance metric for the same simulation as above. The initial deployment shows the agents are quite close together but immediately expand, the variance reduces as the agents become more evenly distributed.

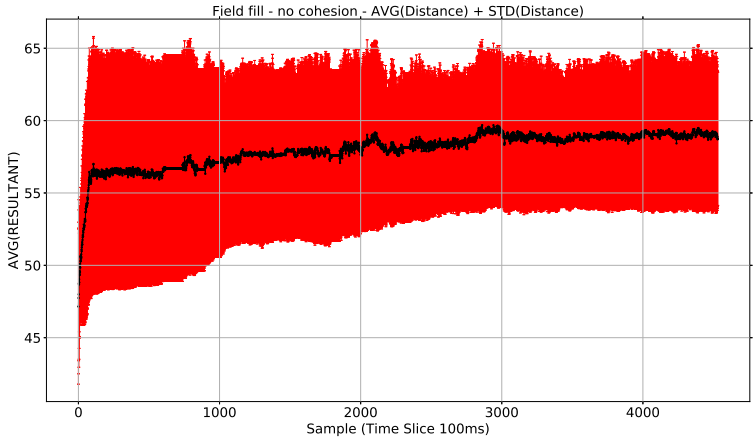


Fig. 7.17: Distance metric 0-450 seconds

Figure 7.18 shows the period when the swarm has initially expanded to a point that the distances appear to stabilise. The stable state is shown as a constant average distance with a constant variance. It cannot be assumed that the agents have stopped moving at this point in the simulation. Agents could change positions such that the average variance is unaffected although this is unlikely. Over this same period the magnitude is zero as shown in Figure 7.13. These two facts therefore allow the swarm's state to be fully realised. The agents are distributed unevenly and are static.

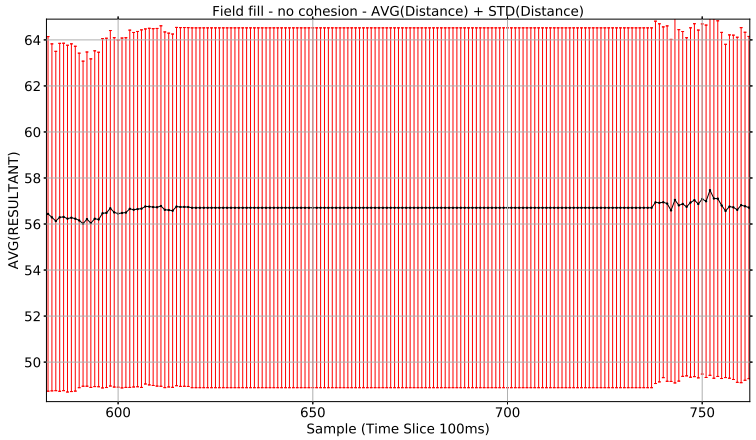


Fig. 7.18: Distance metric 60-75 seconds

Figure 7.19 shows the effect of the first increment of the repulsion field effect increment (50-51 units). The swarm agents move to equalise the *inter-agent vectors* by increasing the distance between the agents. The average distance changes very little due to the irregular distribution of the agents (there is no cohesion to help balance the distribution). The variance in the distance has reduced as the agents are more evenly distributed. At this stage the agents are not filling the space (*inter-agent vector magnitude* has resolved to 0) (Figure 7.15). The two metrics together indicate that the swarm's agent distribution is more even but the swarm is not filling the space.

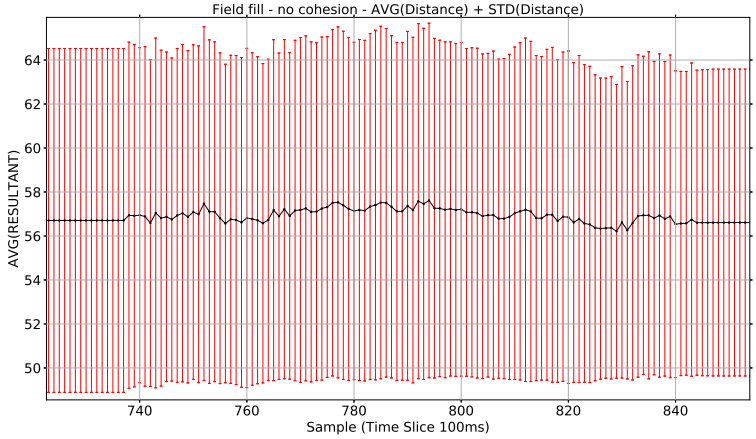


Fig. 7.19: Distance metric 70-85 seconds

Figure 7.20 shows that at each increment the inter-agent average distances varies but the variance decreases as the swarm stabilises and the distribution improves. This improvement in distribution is the agents 'spreading' more evenly as the swarm expands within the area. However after the last field effect increase at 230 seconds the swarm is no longer able to stabilise to a static state.

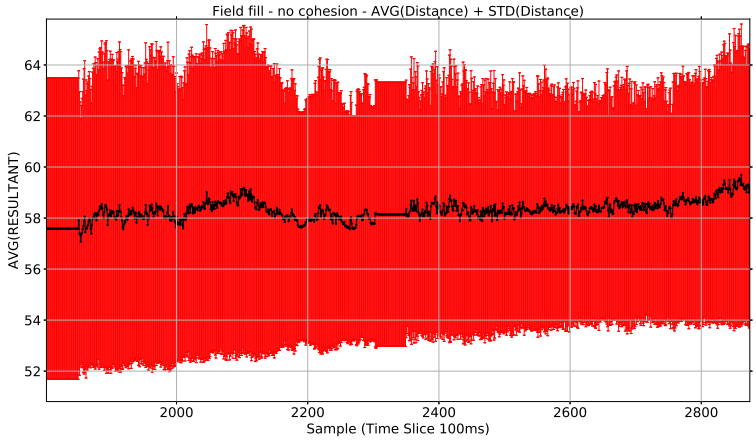


Fig. 7.20: Distance metric 180-280 seconds

Figure 7.21 shows the effect of the final increment. The swarm expands but is unable to stabilise the system. The average distances fluctuate as the agents move to balance the *inter-agent vectors* but the distances are unable to increase any further. The inter-agent distances are effected by the positioning of some of the agents preventing a regular shape to be formed this causes the distance variance to increase and the swarm's structure starts to oscillate. These characteristics indicate that the area may be fully filled. The magnitude at this point is below zero figure 7.16 which indicates expansion of the swarm is possible but not occuring. The conclusion therefore is that the swarm is fully expanded. The magnitude indicates a tendency for the swarm to expand but the distances are not increasing in line with the magnitude increase.

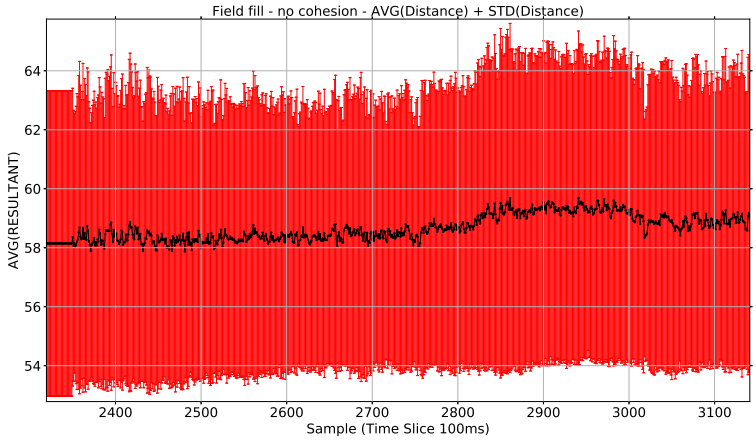


Fig. 7.21: Distance metric 235-315 seconds

7.3 Conclusion

Both of the flood fill techniques work successfully in filling the space. The use of the distance metric is limited in showing the full state state of a swarm due there being no indication of a static state. The *inter-agent vector* metric provides an indication of the ability of the swarm to expand. Using the two metrics together allows the level of the flood filling of a space to be identified. A large distribution in distances indicates possible spaces in the distribution of the agents and ensuring the average *inter-agent vector magnitude* becomes negative ensures the swarm will resolve anomalies in the distribution.

The difference in the approaches of using cohesion combined with repulsion and repulsion only provide two very different characteristics for determining the exit condition for the flood fill. The combination of repulsion and cohesion fields together result in a swarm that moves not only to fill the space but also to accommodate inter-agent relationships as a result the swarm is unable to obtain a static state. Using repulsion only provides clear indications when the space is not filling a space and allows 'gaps' to occur in the swarm's structure as the cohesion is not present even when agents are within visible 'range'.

8. SUMMARY AND ADDITIONAL WORK

8.1 Summary of contributions

In chapter 1 this thesis explores the basic constructs of how animals interact in different types of swarms such as shoals of fish, murmurations of starlings, and ant colonies. The chapter examines how these biological swarms have been studied and modelled to produced autonomous multi-agent robotic systems. The evolution of swarming techniques from the early modelling and mimicking of the biological swarms has shown that metrics are needed to provide an understanding of how agent interactions change over time and how the swarming algorithms affect the resultant swarm structures.

The remainder of the section focuses on the major contributions of the research:

- Model/Simulator
- Inter-agent magnitude metric/Swarm Types
- Perimeter coordination
- Concave Reduction
- Flood filling

8.1.1 Model/Simulator

Earlier research has identified the use of field effects as being an effective method of modelling an agent's behaviour within a swarm. This has led to vector mathematics being the most prominent mathematical modelling tool in swarm theory. Field effect modelling through vector mathematics is discussed in chapter 2. This chapter also introduces a graduated field effect implementation for cohesion (§ 2.4) and repulsion (§ 2.5) this approach helps to reduce collisions. The model also applies an aggregate weighted model that allows the field effect algorithms to be tailored to create regular structures (§ 2.8).

All of the experimental work in this thesis has been carried out using a simulator written specifically for this project. Section 2.13 discusses the simulation process and the object model used for the representation of the swarming agents. The section discusses the mathematical models that have been applied to the simulation. The section also discusses how the simulator is applied to the experiments and how the results are achieved. The simulator has been designed with a specific set of requirements for this thesis and it allows the modelled swarms to be analysed by creating aggregated simulation data sets. The datasets consist of each agent's *inter-agent vector magnitudes*, inter-agent distances, and inter-agent relationships in the form of cohesion and repulsion vectors. The simulator is configurable in terms of the parameters of the weighted model and field effects defined in chapter 2, and for small to medium-sized swarms, allows the effects of varying parameter values to be visualised in real-time. The simulator, being based on an object model, supports two related applications: a graphical environment for the creation of simulation configurations and the visualisation of small-scale experiments, and a command line based application for the execution of large-scale experiments. The simulator uses a discrete time model to capture and implement the models (§ 2.13.3).

8.1.2 Inter-agent magnitude metric

The application of swarms as a platform to solve problems has necessitated the need to understand how agents in a swarm can be distributed and how their movements can be coordinated efficiently. Chapter 3 identifies the *inter-agent vector magnitude* as being a suitable metric to measure a swarm's internal movements (§ 3.5).

The chapter compares this new metric with the distance metric (§ 3.6) currently in use and demonstrates how the *inter-agent vector magnitude* is better suited to identifying the state of a swarm.

The *inter-agent vector* metric is based upon the component vectors that determine the inter-agent interactions rather than the resultant distribution of the agents. This change of focus creates a transferable metric that allows the effect of any field effect based algorithm to be analysed independently of the resultant distributions that the field effects create. The new metric also allows the 'hidden potential' of a swarm to be identified. Inter-agent distance distribution does not show the underlying 'potential' with respect to an agent's current vector magnitude state. The new metric allows a swarm to be identified as being in a repulsive state such that it will expand over time or as being in a cohesive state which results in the swarm remaining a single entity

with a tendency to 'stick' together. The application and benefits of the new metric are discussed further in § 8.2.

In chapter 4 the effect of varying the field effects of a swarm's agents and the impact this has on the swarm's structure is examined. The chapter demonstrates how the field effects can be used to produce different types of swarms based upon inter-agent 'visibility'. The chapter demonstrates how the swarm models can be analysed using both the distance metric and the new *inter-agent vector metric* defined in chapter 3. The chapter demonstrates how identifying the underlying inter-agent interactions allows the type of swarm to be identified and also highlights improvements that can be made in the field effects model to improve a swarm's structure.

8.1.3 Perimeter coordination

If the application of a swarm requires it to move in a specific direction then an additional attribute must be added to the field effect model to create a goal based characteristic. In chapter 5 alternative swarm co-ordination algorithms are presented to create goal-based swarms capable of being applied to reconnaissance type tasks. The alternate methods are compared to a baseline static swarm to identify the effects the algorithms have upon a swarm's structure. The comparisons are carried out using both the distance-based metric and the new *inter-agent vector magnitude* metric. This thesis explores three basic approaches to implementing a goal based characteristic. All agents using their GPS modules, only perimeter based agents using their GPS modules, and finally using a subset of the perimeter selected using a simple counting technique. The findings in this chapter are that by reducing the number of coordinator agents in a swarm it is possible to maintain a stable internal structure whilst imposing a directional bias to a swarm. The most effective technique is to use the smallest subset of the perimeter by using the basic count algorithm (§ 5.8.1). The algorithm that produced the most internal disturbance is the 'all agents' algorithm where all agents are coordinators. The chapter also examines the effects of balancing the field effects on a pro-rata basis (number of coordinators) to reduce the negative effects the algorithms can introduce. This is achieved by adjusting the individual vector component weightings for each of the algorithms used (§ 5.8.5). When the *weighting movement direction* parameters are altered the basic count mechanism is still the most effective algorithm (Figure 5.37).

8.1.4 Concave reduction

Chapter 6 examines how swarms can exhibit emergent behaviours from a simple algorithm. A localised algorithm is used to improve inter-agent distributions by identifying gaps between an agent's neighbours and moving the agent towards the gap. This simple change to the swarming model produces two behavioural changes. It causes a swarm to close voids and migrate towards a more uniform shape. These effects can be interpreted as a 'healing' effect as they cause the swarm to be more uniform in shape with no internal anomalies. This can be useful when a swarm's structure is disrupted by a failed agent or from disruptions to a swarm's path. This behaviour can also be applied to encapsulating an object. The object encapsulation effect is presented in terms of an existing application area in the petrochemical industry where they have identified the use of swarms as a potential method of controlling oil spillages (§ 6.6). The technique used in this thesis uses a Boid-based swarming approach which differs to the current research which uses an ant-colony-based approach. This 'healing' behaviour is also applied to goal-based swarms in an attempt to remove voids in a reconnaissance based swarm when a swarm's path is disrupted (§ 6.7). The thesis demonstrates that the emergent behaviour is able to remove a void created by an obstacle improving the coverage of the swarm's path using the *concave-reduction* effect.

8.1.5 Flood filling

In chapter 7 the cohesion and repulsion field effects (*interaction vectors*) are used to create an area filling behaviour using a swarm of a fixed size. This is achieved by increasing the repulsion field effect of the agents to influence the distribution of the agents such that the agents fill a bounded area. The chapter examines two different techniques of coordinating the expansion process. The first technique involves using a 'normal' swarm that uses both cohesion and repulsion to create a swarm that acts as a single entity and ensures agents maintain 'visibility' of each of its neighbours. The second technique exploits the fact that the swarm is in a bounded area and removes the cohesion field effect and uses only the repulsion field to expand the agents throughout the space. The metrics discussed in chapter 3 are used to identify the effects of the expansion on the swarm's internal structures and to identify terminating conditions for the filling behaviours.

8.2 Future work

The research in this thesis has investigated several issues with respect to swarm analysis but has also raised further questions that need to be answered. The new metric allows swarms to be analysed in a completely different way and has opened up the possibility of analysing new swarm configurations.

8.2.1 Magnitude metric application

Swarms are often constructed from heterogeneous agents that adapt to stimuli but the agents themselves are usually modelled using homogeneous field effects. This modelling of homogeneous field effects lends itself to being measured and monitored by a distance based metric as discussed by Navarro et al. [104] and Gazi et al. ([130, 37, 39, 40, 38, 41, 42]) due to the regular shapes and structures that emerge. The dependence on regularity of shapes and structures is a limitation of the distance metric due to the aggregation of the distances not reflecting the mathematical model that creates the structures. The magnitude based metric overcomes that limitation by analysing the vectors that effect the agent distribution. With heterogeneous field effects the inter-agent distances may vary in an equilibrium state due to the way the field effects overlap when agents interact.

The *inter-agent vector magnitude* metric takes into consideration the resultant effects of the mathematical model rather than just arbitrary distances between agents. The balancing of *interaction vectors* affect the resultant agent distributions.

The additional work required here would be to identify potential applications of these variations in the field effects that could improve existing swarm applications. For example, if an agent is on a perimeter would the swarm benefit from reducing the field effects to create a greater compression effect? Would having a mixed field effect swarm produce better coverage of an area? What would be the impact of having variable field effects upon a goal-based swarm?

8.2.2 Area flooding

If agents could vary their field effects based on their position in a swarm (e.g. agents at the boundary) is it possible to improve coverage?

If agents were able to vary their field effects based on detected environmental features a swarm may be able to achieve set goals more efficiently in unknown environments. An

environment may have narrow paths linking larger chambers. Allowing agents to change their field effects may allow a swarm to propagate through this type of environment more effectively. This application requires further work to identify the effectiveness of using adaptive field effects on area flooding.

8.2.3 Path following and shape forming swarms

The *destination vector* as discussed in § 2.7 has been based upon a swarm having a fixed single destination however this model can be changed slightly to produce two more possible swarming effects by applying the *destination vector* to a set of points. The effects that are possible are to create swarms that form arbitrary shapes or a swarm that will migrate along a specific path. The *destination vector* required to achieve these two emergent behaviours can be implemented using the coordination techniques discussed in chapter 5.

The main area of research that is still required in this area is how to best implement the coordination. The thesis shows that changing how coordination is applied via perimeter coordinators can improve a swarm's structure however what is not known is what is the best way to apply the coordination if a swarm is being applied to a shape forming [31] or path following task.

8.2.4 Self optimisation

The research carried out in terms of improving the performance of a swarm's internal distribution shows that as the parameters are changed it is possile to improve the performance of a swarm. Further work is therefore required to identify what optimisation techniques could be applied to a swarm's algorithms to improve its performance based on environmental conditions.

BIBLIOGRAPHY

[1] J. Abouaf. Trial by fire: Teleoperated robot targets Chernobyl. *IEEE Computer Graphics and Applications*, 18(4):10–14, Jul 1998.

[2] M. S. Alvissalim, B. Zaman, Z. A. Hafizh, M. A. Ma'sum, G. Jati, W. Jatmiko, and P. Mursanto. Swarm quadrotor robots for telecommunication network coverage area expansion in disaster area. In *Proceedings of SICE Annual Conference (SICE)*, pages 2256–2261, Aug 2012.

[3] Panayiotis Andreou, Demetrios Zeinalipour-Yazti, Maria Andreou, Panos K Chrysanthis, and George Samaras. Perimeter-based data replication in mobile sensor networks. In *Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, (MDM'09).*, pages 244–251. IEEE, 2009.

[4] Panayiotis Andreou, Demetrios Zeinalipour-Yazti, Panos K Chrysanthis, and George Samaras. In-network data acquisition and replication in mobile sensor networks. *Distributed and Parallel Databases*, 29(1-2):87–112, 2011.

[5] Ronald C Arkin, Tucker Balch, and Elizabeth Nitz. Communication of behavorial state in multi-agent retrieval tasks. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 588–594. IEEE, 1993.

[6] Shinichi Aso, Sho Yokota, Hiroshi Hashimoto, Yasuhiro Ohyama, Akinori Sasaki, and Hiroaki Kobayashi. Control and stability for robotic swarm based on center of gravity of local swarm. In *IEEE International Symposium on Industrial Electronics*, pages 1341–1346. IEEE, 2008.

[7] F. Augugliaro, E. Zarfati, A. Mirjan, and R. D'Andrea. Knot-tying with flying machines for aerial construction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5917–5922, Sept 2015.

[8] A. Banharnsakun, T. Achalakul, and R. C. Batra. Target finding and obstacle avoidance algorithm for microrobot swarms. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1610–1615, Oct 2012.

[9] Jan Carlo Barca and Y Ahmet Sekercioglu. Swarm robotics reviewed. *Robotica*, 31(03):345–359, 2013.

[10] Laura Barnes, Wendy Alvis, MaryAnne Fields, Kimon Valavanis, and Wilfrido Moreno. Heterogeneous swarm formation control using bivariate normal functions to generate potential fields. In *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, pages 85–94. IEEE, 2006.

[11] Laura Barnes, Wendy Alvis, MaryAnne Fields, Kimon Valavanis, and Wilfrido Moreno. Swarm formation control with potential fields formed by bivariate normal functions. In *Control and Automation, 2006. MED'06. 14th Mediterranean Conference on*, pages 1–7. IEEE, 2006.

[12] Laura Barnes, MaryAnne Fields, and Kimon Valavanis. Unmanned ground vehicle swarm formation control using potential fields. In *Mediterranean Conference on Control & Automation (MED'07).*, pages 1–8. IEEE, 2007.

[13] Laura E Barnes, Mary Anne Fields, and Kimon P Valavanis. Swarm formation control utilizing elliptical surfaces and limiting functions. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(6):1434–1445, 2009.

[14] Derek J. Bennet and C.R. McInnes. Verifiable control of a swarm of unmanned aerial vehicles. *Journal of Aerospace Engineering*, 223(7):939–953, 2009.

[15] Spring Berman, Adám Halász, Vijay Kumar, and Stephen Pratt. Bio-inspired group behaviors for the deployment of a swarm of robots to multiple destinations. In *IEEE International Conference on Robotics and Automation*, pages 2318–2323. IEEE, 2007.

[16] The british beekeepers association. Life in the hive. `http://www.bbka.org.uk/learn/general_information/life_in_the_hive`, ND. [Online; accessed 19-Apr-2016].

[17] Matthias R Brust and Bogdan M Strimbu. A networked swarm model for UAV deployment in the assessment of forest environments. In *IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6. IEEE, 2015.

[18] Ning CAI, Hai-Ying MA, Ming-Hua LIU, and Cai-Xia WANG. Swarm stability analysis of nonlinear time-varying heterogeneous dynamical multi-agent systems. In *Proceeding of the 31st Chinese Control Conference*, pages 6292–6295. IEEE, 2012.

[19] B. D. Campbell and F. Samsel. Murmurations: Drawing together art, visualization, and physical phenomena. *IEEE Computer Graphics and Applications*, 35(4):8–12, July 2015.

[20] Andrea Cavagna, Alessio Cimarelli, Irene Giardina, Giorgio Parisi, Raffaele Santagati, Fabio Stefanini, and Massimiliano Viale. Scale-free correlations in starling flocks. *Proceedings of the National Academy of Sciences*, 107(26):11865–11870, 2010.

[21] F. A. Cheein, D. Herrera, J. Gimenez, R. Carelli, M. Torres-Torriti, J. R. Rosell-Polo, A. Escol, and J. Arn. Human-robot interaction in precision agriculture: Sharing the workspace with service units. In *IEEE International Conference on Industrial Technology (ICIT)*, pages 289–295, March 2015.

[22] B. Chu. Mobile robot position control algorithm based on multiple ultrasonic distance sensors. In *15th International Conference on Control, Automation and Systems (ICCAS)*, pages 1238–1240, Oct 2015.

[23] Oracle Corporation. MySQL. `http://www.mysql.com`, 2016. [Online; accessed 15-Apr-2016].

[24] Nikolaus Correll and Daniela Rus. Architectures and control of networked robotic systems. *Handbook of collective robotics: fundamentals and challenges*, pages 81–103, 2013.

[25] X. Cui, C. T. Hardin, R. K. Ragade, and A. S. Elmaghraby. A swarm approach for emission sources localization. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 424–430, Nov 2004.

[26] Y. S. Dai, M. Hinchey, M. Madhusoodan, J. L. Rash, and X. Zou. A prototype model for self-healing and self-reproduction in swarm robotics system. In *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 3–10, Sept 2006.

[27] G. N. Desouza and A. C. Kak. Vision for mobile robot navigation: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):237–267, Feb 2002.

[28] Georgi Dinolov. *Swarm Control Through Symmetry and Distribution Characterization*. Harvey Mudd College, May 2011.

[29] Marco Dorigo. Swarm-bots and swarmanoid: Two experiments in embodied swarm intelligence. In *Web intelligence*, pages 2–3, 2009.

[30] Marco Dorigo, Elio Tuci, Roderich Groß, Vito Trianni, Thomas Halva Labella, Shervin Nouyan, Christos Ampatzis, Jean-Louis Deneubourg, Gianluca Baldassarre, Stefano Nolfi, Francesco Mondada, Dario Floreano, and Luca Maria Gambardella. The swarm-bots project. In *Proceedings of the International Conference on Swarm Robotics*, SAB'04, pages 31–44, Berlin, Heidelberg, 2005. Springer-Verlag.

[31] S. W. Ekanayake and P. N. Pathirana. Geometric formations in swarm aggregation: An artificial formation force based approach. In *Third International Conference on Information and Automation for Sustainability (ICIAFS 2007).*, pages 82–87, Dec 2007.

[32] Samitha W Ekanayake and Pubudu N Pathirana. Formations of robotic swarm: an artificial force based approach. *International Journal of Advanced Robotic Systems*, 7(3):173–190, 2010.

[33] SparkFun Electronics. Sparkfun venus GPS with SMA connector. `https://www.sparkfun.com/products/11058`, September 2016. [Online; accessed 24-Sept-2016].

[34] Anders Eriksson, Martin Nilsson Jacobi, Johan Nyström, and Kolbjørn Tunstrøm. Determining interaction rules in animal swarms. *Oxford University Press : Behavioral Ecology*, 21(5):1106–1111, 2010.

[35] Lei Fang and Panos J Antsaklis. Information consensus of asynchronous discrete-time multi-agent systems. In *Proceedings of the American Control Conference*, pages 1883–1888. IEEE, 2005.

[36] Python Software Foundation. cymysql 0.8.5 : Python package index. `https://pypi.python.org/pypi/cymysql`, 2015. [Online; accessed 15-Apr-2016].

[37] V. Gazi. Swarm aggregations using artificial potentials and sliding-mode control. *IEEE Transactions on Robotics*, 21(6):1208–1214, Dec 2005.

[38] V. Gazi and K. M. Passino. Stability analysis of social foraging swarms. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):539–557, Feb 2004.

[39] Veysel Gazi and Kevin M Passino. Stability analysis of swarms in an environment with an attractant/repellent profile. In *Proceedings of the 2002 American Control Conference*, volume 3, pages 1819–1824. IEEE, 2002.

[40] Veysel Gazi and Kevin M. Passino. A class of attractions/repulsion functions for stable swarm aggregations. *International Journal of Control*, 77(18):1567–1579, 2004.

[41] Veysel Gazi and Kevin M Passino. Stability of a one-dimensional discrete-time asynchronous swarm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(4):834–841, 2005.

[42] Veysel Gazi and Kevin M Passino. *Swarm stability and optimization*. Springer Science & Business Media, 2011.

[43] Google. Google launches Project Loon. `http://ravenaerostar.com/products/high-altitude-ballons/project-loon-balloons`. [Online; accessed 30-Sept-2014].

[44] Google. Project Loon Homepage. `https://www.google.com/loon/`, 2013. [Online; accessed 18-Jan-2014].

[45] Erico Guizzo. video: watch flying robots build a 6 meter tower - IEEE Spectrum. `http://spectrum.ieee.org/automaton/robotics/diy/video-watch-flying-robots-build-a-6-meter-tower`, Dec 2011. [Online; accessed 1-Apr-2016].

[46] Hongliang Guo, Yan Meng, and Yaochu Jin. Swarm robot pattern formation using a morphogenetic multi-cellular based self-organizing algorithm. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3205–3210. IEEE, 2011.

[47] Pini Gurfil and Elad Kivelevitch. Flock properties effect on task assignment and formation flying of cooperating unmanned aerial vehicles. *Proceedings of the*

Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 221(3):401–416, 2007.

[48] Vishwesha Guttal, Pawel Romanczuk, Stephen J. Simpson, Gregory A. Sword, and Iain D. Couzin. Cannibalism can drive the evolution of behavioural phase polyphenism in locusts. *Journal of Ecology Letters*, 15(10):1158–1166, 2012.

[49] S. Gler, N. Kksal, and B. Fidan. Adaptive control of a three-agent surveillance swarm with constant speed constraint. In *2013 9th Asian Control Conference (ASCC)*, pages 1–6, June 2013.

[50] T. C. Hales. The Honeycomb Conjecture. *Discrete & Computational Geometry*, 25(1):1–22, 2001.

[51] H. Hamann, J. Stradner, T. Schmickl, and K. Crailsheim. A hormone-based controller for evolutionary multi-modular robotics: From single modules to gait learning. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, July 2010.

[52] Heiko Hamann and Heinz Wörn. A framework of space–time continuous models foralgorithm design in swarm robotics. *Swarm Intelligence*, 2(2):209–239, 2008.

[53] Hiroshi Hashimoto, Shinichi Aso, Syo Yokota, Akinori Sasaki, Yasuhiro Ohya, and Hiroyuki Kobayashi. Stability of swarm robot based on local forces of local swarms. In *SICE Annual Conference*, pages 1254–1257. IEEE, 2008.

[54] New Zealand Herald. Google launches Project Loon. `http://www.nzherald.co.nz/nz/news/article.cfm?c_id=1&objectid=10890750`, June 2013. [Online; accessed 30-Sept-2014].

[55] J. Hereford. Analysis of beeclust swarm algorithm. In *IEEE Symposium on Swarm Intelligence (SIS)*, pages 1–7, April 2011.

[56] F. Higgins, A. Tomlinson, and K. M. Martin. Survey on security challenges for swarm robotics. In *Fifth International Conference on Autonomic and Autonomous Systems (2009 ICAS)*, pages 307–312, April 2009.

[57] M. G. Hinchey, R. Sterritt, and C. Rouff. Swarms and swarm intelligence. *Computer*, 40(4):111–113, April 2007.

[58] Nicholas Hoff, A. Sagoff, Robert J. Wood, and Radhika Nagpal. Two foraging algorithms for robot swarms using only local communication. In *IEEE International Conference on Robotics and Biomimetics, ROBIO 2010, Tianjin, China, December 14-18, 2010*, pages 123–130, 2010.

[59] Nicholas Hoff, Robert Wood, and Radhika Nagpal. Effect of sensor and actuator quality on robot swarm algorithm performance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4989–4994. IEEE, 2011.

[60] Johnny Holmström and Daniel Romero. A survey of robotic swarms. In *Conference on Interesting Results in Computer Science and Engineering (IRCSE'10)*. Mälardalen University, Sweden, 2010.

[61] S. P. Hou and C. C. Cheah. Multiplicative potential energy function for swarm control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS 2009)*, pages 4363–4368, Oct 2009.

[62] S. P. Hou, C. C. Cheah, and J. J. E. Slotine. Dynamic region following formation control for a swarm of robots. In *IEEE International Conference on Robotics and Automation (ICRA '09).*, pages 1929–1934, May 2009.

[63] John Hunter. Matplotlib: Python plotting - Matplotlib 1.5.1 Documentation. `http://matplotlib.org/`, February 2016. [Online; accessed 15-Apr-2016].

[64] O Ilaya, C Bil, and M Evans. Control design for unmanned aerial vehicle swarming. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 222(4):549–567, 2008.

[65] Luca Iocchi, Daniele Nardi, and Massimiliano Salerno. *Balancing Reactivity and Social Deliberation in Multi-Agent Systems: From RoboCup to Real-World Applications*, chapter Reactivity and Deliberation: A Survey on Multi-Robot Systems, pages 9–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[66] Amelia Ritahani Ismail and Jon Timmis. Towards self-healing swarm robotic systems inspired by granuloma formation. In *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 313–314. IEEE, 2010.

[67] James D. McLurkin IV. *Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems*. PhD thesis, Massachusetts Institute of Technology, USA, 2008.

[68] B. Jakimovski, B. Meyer, and E. Maehle. Swarm intelligence for self-reconfiguring walking robot. In *IEEE Swarm Intelligence Symposium*, pages 1–8, Sept 2008.

[69] Raphaël Jeanson, Francis L W Ratnieks, and Jean-Louis Deneubourg. Pheromone trail decay rates on different substrates in the Pharaoh's ant, Monomorium Pharaonis (L.). *Physiological Entomology*, 28(3):192–198, 2003.

[70] Shin-Young Jung and Michael A Goodrich. Multi-robot perimeter-shaping through mediator-based swarm control. In *16th International Conference on Advanced Robotics (ICAR)*, pages 1–6. IEEE, 2013.

[71] Kanchan Kamnani and Chaitali Suratkar. A review paper on Google Loon technique. *International Journal of Research In Science and Engineering*, 1(Special Issue:1):167–171, 2016.

[72] H. Kawabayashi and Y. W. Chen. Interactive system of artificial fish school based on the extended boid model. In *International Conference on Intelligent Information Hiding and Multimedia Signal Processing (2008 IIHMSP)*, pages 721–724, Aug 2008.

[73] Morgan Kelly. Army ants living bridges span collective intelligence, swarm robotics (pnas). `https://blogs.princeton.edu/research/2015/11/24/army-ants-living-bridges-span-collective-intelligence-swarm-robotics-pnas/`, November 2015. [Online; accessed 13-Apr-2016].

[74] Naga K. C. Krothapalli and Abhijit V. Deshmukh. Design of negotiation protocols for multi-agent manufacturing systems. *International journal of production research*, 37(7):1601–1624, 1999.

[75] M. Kudelski, M. Cinus, L. Gambardella, and G. A. Di Caro. A framework for realistic simulation of networked multi-robot systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5018–5025, Oct 2012.

[76] Vijay Kumar. Current projects: Vijay Kumar Lab. `http://www.kumarrobotics.org/research/`, 2016. [Online; accessed 21-Jan-2016].

[77] Geunho Lee and Nak Young Chong. Self-configurable mobile robot swarms with hole repair capability. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (2008 IROS).*, pages 1403–1408, Sept 2008.

[78] J. H. Lee, C. W. Ahn, and J. An. A honey bee swarm-inspired cooperation algorithm for foraging swarm robots: An empirical analysis. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 489–493, July 2013.

[79] S. W. Lee, H. Jeong, S. K. Lee, Young-Kweon Kim, and J. H. Park. Lidar system using indirect time of flight method and mems scanner for distance measurement. In *International Conference on Optical MEMS and Nanophotonics (OMN)*, pages 1–2, July 2016.

[80] Bin Lei, Wenfeng Li, and Fan Zhang. Stable flocking algorithm for multi-robot systems formation control. In *IEEE Congress on Evolutionary Computation. (CEC 2008)*, pages 1544–1549. IEEE, 2008.

[81] Ming Li, Anthony Alvarez, Francesco De Pellegrini, B Prabhakaran, and Imrich Chlamtac. Robotrak: a centralized real-time monitoring, control, and coordination system for robot swarms. In *Proceedings of the 1st international conference on Robot communication and coordination*, page 37. IEEE Press, 2007.

[82] Aleksis Liekna and J Grundspenkins. Towards practical application of swarm robotics: overview of swarm tasks. In *Proceedings of the 13th International Conference Engineering for Rural Development, Jelgava, Latvia*, 2014.

[83] Bo Liu, Tianguang Chu, and Long Wang. Stability and oscillation of swarm with interaction time delays. In *American Control Conference (ACC'07)*, pages 4600–4605. IEEE, 2007.

[84] Wenguo Liu and Alan Winfield. Modelling and optimisation of adaptive foraging in swarm robotic systems. *The International Journal of Robotics Research*, 2010.

[85] Wenguo Liu, Alan FT Winfield, and Jin Sa. Modelling swarm robotic systems: A case study in collective foraging. *Towards autonomous robotic systems (TAROS 07)*, pages 25–32, 2007.

[86] CYBERBOTICS Ltd. Webots:webots. `https://www.cyberbotics.com/webots.php`, February 2016. [Online; accessed 15-Apr-2016].

[87] Z. Ma and G. A. E. Vandenbosch. Comparison of weighted sum approaches for PSO fitness functions in antenna design. In *42nd European Microwave Conference (EuMC)*, pages 842–845, Oct 2012.

[88] A. E. Magurran and T. J. Pitcher. Provenance, shoal size and the sociobiology of predator-evasion behaviour in minnow shoals. *Proceedings of the Royal Society of London B: Biological Sciences*, 229(1257):439–465, 1987.

[89] Marco Mamei, Matteo Vasirani, and Franco Zambonelli. Experiments of morphogenesis is swarms of simple mobile robots. *Applied Artificial Intelligence*, 18(9-10):903–919, 2004.

[90] Nithin Mathews, Gabriele Valentini, Anders Lyhne Christensen, Rehan O'Grady, Arne Brutschy, and Marco Dorigo. Spatially targeted communication in decentralized multirobot systems. *Autonomous Robots*, 38(4):439–457, 2015.

[91] MathWorks Inc. MATLAB - Mathworks - Mathworks United Kingdom. `http://uk.mathworks.com/products/matlab/`, 1994. [Online; accessed 09-May-2016].

[92] James McLurkin and Erik D Demaine. A distributed boundary detection algorithm for multi-robot systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*, pages 4791–4798. IEEE, 2009.

[93] Don Miner. Swarm robotics algorithms: A survey. *Report, MAPLE lab, University of Maryland*, 2007.

[94] Don Miner and Niels Kasch. Swarmvis: a tool for visualizing swarm systems. *UMBC Computer Science*, 636, 2011.

[95] Yogeswaran Mohan and SG Ponnambalam. An extensive review of research in swarm robotics. In *World Congress on Nature & Biologically Inspired Computing (2009 NaBIC)*, pages 140–145. IEEE, 2009.

[96] F. Mondada, L. M. Gambardella, D. Floreano, S. Nolfi, J. L. Deneuborg, and M. Dorigo. The cooperation of swarm-bots: physical interactions in collective robotics. *IEEE Robotics Automation Magazine*, 12(2):21–28, June 2005.

[97] Francesco Mondada, André Guignard, Michael Bonani, Daniel Bär, Michel Lauria, and Dario Floreano. Swarm-bot: From concept to implementation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.(IROS 2003)*, volume 2, pages 1626–1631. IEEE, 2003.

[98] Luc Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, 50(2):169–182, 2005.

[99] Y. Mulgaonkar, G. Cross, and V. Kumar. Design of small, safe and robust quadrotor swarms. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2208–2215, May 2015.

[100] Purushotham Muniganti and Albert Oller Pujol. A survey on mathematical models of swarm robotics. In *Workshop of physical agents*, pages 29–30, 2010.

[101] D. Salda na, D. Ovalle, and A. Montoya. Improved algorithm for perimeter tracking in robotic sensor networks. In *Conferencia Latinoamericana En Informatica (CLEI)*, pages 1–7, Oct 2012.

[102] Alex Nash and Sven Koenig. Lazy theta*: Any-angle path planning and path length analysis in 3d. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010.

[103] Iñaki Navarro, Álvaro Gutiérrez, Fernando Matía, and Félix Monasterio-Huelin. An approach to flocking of robots using minimal local sensing and common orientation. In *Hybrid Artificial Intelligence Systems*, pages 616–624. Springer, 2008.

[104] Inaki Navarro and F Matía. A proposal of a set of metrics for collective movement of robots. *Proc. Workshop on Good Experimental Methodology in Robotics*, 2009.

[105] Iñaki Navarro and Fernando Matía. An introduction to swarm robotics. *ISRN Robotics*, 2013, 2012.

[106] CBS News. Google's ambitious internet balloons soar above New Zealand. `http://www.cbsnews.com/news/googles-ambitious-internet-balloons-soar-above-new-zealand/`, 2013. [Online; accessed 30-Sept-2014].

[107] Reza Olfati-Saber, Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[108] Randal Olson. Current Research — Dr. Randal S. Olson. `http://www.randalolson.com/about/research/`, April 2016. [Online; accessed 13-Apr-2016].

[109] Fuchen Pan, Xue-Bo Chen, and Lin Li. Practical stability analysis of stochastic swarms. *The 3rd Intetnational Conference on Innovative Computing Information and Control (ICICIC'08)*, 2008.

[110] Fuchen Pan, Xuebo Chen, and Lin Li. Practical stability in swarms system. *Journal of applied Math & Informatics*, 26(1-2):203–212, 2008.

[111] Weiyun Pan and Y. Zheng. Stability analysis of general swarm with gaussian attractant/repellent profiles and interaction time delays. In *32nd Chinese Control Conference (CCC)*, pages 1224–1229, July 2013.

[112] Daniel J. G. Pearce, Adam M. Miller, George Rowlands, and Matthew S. Turner. Role of projection in the control of bird flocks. *Proceedings of the National Academy of Sciences*, 111(29):10422–10426, 2014.

[113] R Pedrami and BW Gordon. Control and cohesion of energetic swarms. In *American Control Conference*, pages 129–134. IEEE, 2008.

[114] Jacques Penders, Lyuba Alboul, Ulf Witkowski, Amir Naghsh, Joan Saez-Pons, Stefan Herbrechtsmeier, and Mohamed El-Habbal. A robot swarm assisting a human fire-fighter. *Advanced Robotics*, 25(1-2):93–117, 2011.

[115] Duc Truong Pham and M Castellani. The bees algorithm: modelling foraging behaviour to solve continuous optimization problems. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 223(12):2919–2938, 2009.

[116] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, T. Stirling, Gutirrez, L. M. Gambardella, and M. Dorigo. Argos: A modular, multi-engine simulator for heterogeneous swarm robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5027–5034, Sept 2011.

[117] Marios M Polycarpou, Yanli Yang, and Kevin M Passino. Cooperative control of distributed multi-agent systems. *IEEE Control Systems Magazine*, 21:1–27, 2001.

[118] Oliver Purwin and Raffaello D'Andrea. Trajectory generation and control for four wheeled omnidirectional vehicles. *Robotics and Autonomous Systems*, 54(1):13 – 22, 2006.

[119] Python Software Foundation. Welcome to python.org. `http://www.python.org/`, 2015. [Online; accessed 21-Jan-2015].

[120] Long Qin, Yabing Zha, Quanjun Yin, and Yong Peng. Formation control of robotic swarm using bounded artificial forces. *The Scientific World Journal*, 2013, 2013.

[121] R. Ramaithitima, M. Whitzer, S. Bhattacharya, and V. Kumar. Sensor coverage robot swarms using local sensing without metric information. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3408–3415, May 2015.

[122] R. Ramaithitima, M. Whitzer, S. Bhattacharya, and V. Kumar. Automated creation of topological maps in unknown environments using a swarm of resource-constrained robots. *IEEE Robotics and Automation Letters*, 1(2):746–753, July 2016.

[123] Tim Räz. On the Application of the Honeycomb Conjecture to the Bee's Honeycomb. *Philosophia Mathematica*, page nkt022, 2013.

[124] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH computer graphics*, volume 21:4, pages 25–34. ACM, 1987.

[125] Lorenzo Riano and Martin Mcginnity. Emergent behaviours at the edge of chaos, 2011.

[126] J. H. Roach, R. J. Marks, and B. B. Thompson. Recovery from sensor failure in an evolving multiobjective swarm. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):170–174, Jan 2015.

[127] EJH Robinson, KE Green, EA Jenner, M Holcombe, and FLW Ratnieks. Decay rates of attractive and repellent pheromones in an ant foraging trail network. *Insectes sociaux*, 55(3):246–251, 2008.

[128] Ivo Roghair, Martin Van Sint Annaland, and Hans J. A. M. Kuipers. Drag force and clustering in bubble swarms. *AIChE Journal*, 59(5):1791–1800, 2013.

[129] M. Rubenstein and W. M. Shen. A scalable and distributed approach for self-assembly and self-healing of a differentiated shape. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008)*, pages 1397–1402, Sept 2008.

[130] J Saez-Pons, Lyuba Alboul, Veysel Gazi, and Jacques Penders. Non-communicative robot swarming in the guardians project. *Proceedings of the EURON/IARP International Workshop on Robotics for Risky Interventions and Surveillance of the Environment., Benicassim, Spain*, 2008.

[131] Héctor F. Satizábal, Andres Upegui, Andres Perez-Uribe, Philippe Rétornaz, and Francesco Mondada. A social approach for target localization: simulation and implementation in the marxbot robot. *Memetic Computing*, 3(4):245–259, 2011.

[132] M. Scheutz and P. Bauer. A scalable, robust, ultra-low complexity agent swarm for area coverage and interception tasks. In *IEEE Conference on Computer Aided Control System Design, IEEE International Conference on Control Applications, IEEE International Symposium on Intelligent Control*, pages 1258–1263, Oct 2006.

[133] F. E. Schneider and D. Wildermuth. A potential field based approach to multi robot formation navigation. In *IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*, volume 1, pages 680–685 vol.1, Oct 2003.

[134] A. M. Schroeder and M. Kumar. Design of decentralized chemotactic control law for area coverage using swarm of mobile robots. In *American Control Conference (ACC)*, pages 4317–4322, July 2016.

[135] Thomas D Seeley, P Kirk Visscher, and Kevin M Passino. Group decision making in honey bee swarms. *American Scientist*, Volume 94, 2006.

[136] SEUL. Pygame. `http://www.pygame.org/`, 2015. [Online; accessed 21-Jan-2015].

[137] Jeff S Shamma. *Cooperative control of distributed multi-agent systems*. Wiley Online Library, 2007.

[138] Wei-Min Shen, Peter Will, Aram Galstyan, and Cheng-Ming Chuong. Hormone-inspired self-organization and distributed control of robotic swarms. *Autonomous Robots*, 17(1):93–105, July 2004.

[139] Z. Shi, J. Tu, Q. Zhang, X. Zhang, and J. Wei. The improved q-learning algorithm based on pheromone mechanism for swarm robot system. In *32nd Chinese Control Conference (CCC)*, pages 6033–6038, July 2013.

[140] Zhiguo Shi, Jun Tu, Junming Wei, Qiao Zhang, and Xiaomeng Zhang. The simulation scenario for swarm robots based on open-source software player/stage. In *International Workshop on Open-Source Software for Scientific Computation (OSSC)*, pages 107–113, Oct 2011.

[141] David Smalley. Autonomous Warfare - A Revolution in Military Affairs. `http://www.onr.navy.mil/Media-Center/Press-Releases/2015/`

LOCUST-low-cost-UAV-swarm-ONR.aspx, April 2015. [Online; accessed 15-Apr-2016].

[142] J. Song and S. Gupta. SLAM based shape adaptive coverage control using autonomous vehicles. In *10th System of Systems Engineering Conference (SoSE)*, pages 268–273, May 2015.

[143] S. Srivastava and G. C. Nandi. Localization of mobile robots in a network using mobile agents. In *International Conference on Computer and Communication Technology (ICCCT)*, pages 415–420, Sept 2010.

[144] D. J. Stilwell and J. S. Bay. Toward the development of a material transport system using swarms of ant-like robots. In *IEEE International Conference on Robotics and Automation*, pages 766–771 vol.1, May 1993.

[145] D. P. Stormont. Autonomous rescue robot swarms for first responders. In *Proceedings of the IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety (CIHSPS)*, pages 151–157, March 2005.

[146] Ariana Strandburg-Peshkin, Damien R. Farine, Iain D. Couzin, and Margaret C. Crofoot. Shared decision-making drives collective movement in wild baboons. *Science*, 348(6241):1358–1361, 2015.

[147] Ruben Stranders, Francesco Maria Delle Fave, Alex Rogers, and Nick Jennings. A decentralised coordination algorithm for mobile sensors. *Association for the Advancement of Artificial Intelligence (www.aaai.org)*, 2010.

[148] Chad M. Topaz, Maria R. D'Orsogna, Leah Edelstein-Keshet, and Andrew J. Bernoff. Locust dynamics: Behavioral phase change and swarming. *PLoS Comput Biol*, 8(8):1–11, 08 2012.

[149] B. R. Trilaksono. Distributed consensus control of robot swarm with obstacle and collision avoidance. In *2nd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, pages 2–2, Oct 2015.

[150] G. Tuna and K. Gulez. Aided navigation techniques for indoor and outdoor unmanned vehicles. In *5th International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–4, May 2012.

[151] Gabriele Valentini, Heiko Hamann, and Marco Dorigo. Self-organized collective decision-making in a 100-robot swarm. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 4216–4217, 2015.

[152] Maarten Vankerkom and Jin Yu. *Visualizing Swarm Algorithms*. Katholieke Universitiet Leuven, 2004.

[153] E. Vassev and M. Hinchey. ASSL Specification and Code Generation of Self-Healing Behavior for NASA Swarm-Based Systems. In *Sixth IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, pages 77–86, April 2009.

[154] R. R. P. Vicerra, R. N. R. Barcos, J. K. S. Bulan, A. J. O. Loterina, S. O. Oliver, J. M. D. G. Pineda, A. R. dela Cruz, E. A. Roxas, A. A. Bandala, and E. P. Dadios. A comparative study of swarm foraging behaviors; trophallaxis, task allocation and pheromone. In *International Conference on Humanoid, Nanotechnology, Information Technology,Communication and Control, Environment and Management (HNICEM)*, pages 1–6, Dec 2015.

[155] Liangshun Wang and Huajing Fang. Stability analysis of improved local swarms. In *31st Chinese Control Conference (CCC)*, pages 6075–6080. IEEE, 2012.

[156] Maxim Worchester. Autonomous Warfare - A Revolution in Military Affairs. `https://www.files.ethz.ch/isn/190160/340_Worcester.pdf`, April 2015. [Online; accessed 15-Apr-2016].

[157] Naixue Xiong, Jing He, Yan Yang, Yanxiang He, Tai hoon Kim, and Chuan Lin. A survey on decentralized flocking schemes for a set of autonomous mobile robots (invited paper). *Journal of Communications*, 5(1), 2010.

[158] Y. Xue, G. Tian, and B. Huang. Optimal robot path planning based on danger degree map. In *IEEE International Conference on Automation and Logistics (ICAL '09)*, pages 1040–1045, Aug 2009.

[159] B. Yang, Y. Ding, and K. Hao. Area coverage searching for swarm robots using dynamic voronoi-based method. In *34th Chinese Control Conference (CCC)*, pages 6090–6094, July 2015.

[160] Mao Yang, Gan-gui Yan, and Yan-tao Tian. A review of studies in flocking for multi-robot system. In *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE)*, volume 4, pages 28–31. IEEE, 2010.

[161] Y. T. Yao and R. H. Hwang. Analysis of Swarm Behavior of Users' GPS Data Based on Boids Model. In *IEEE International Conference on Internet of Things (iThings), and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing(CPSCom)*, pages 421–427, Sept 2014.

[162] George F. Young, Luca Scardovi, Andrea Cavagna, Irene Giardina, and Naomi E. Leonard. Starling flock networks manage uncertainty in consensus at low cost. *PLoS Comput Biol*, 9(1):1–7, 01 2013.

[163] Demetrios Zeinalipour-Yazti, Panayiotis Andreou, Panos K. Chrysanthis, and George Samaras. Senseswarm: A perimeter-based data acquisition framework for mobile sensor networks. In *Proceedings of the 4th Workshop on Data Management for Sensor Networks: In Conjunction with 33rd International Conference on Very Large Data Bases*, DMSN '07, pages 13–18, New York, NY, USA, 2007. ACM.

[164] C. Zhang, F. m. Zhang, F. Li, and H. s. Wu. Improved artificial fish swarm algorithm. In *IEEE 9th Conference on Industrial Electronics and Applications (ICIEA)*, pages 748–753, June 2014.

[165] Guoxian Zhang, Gregory K Fricke, and Devendra P Garg. Spill detection and perimeter surveillance via distributed swarming agents. *IEEE/ASME Transactions on Mechatronics*, 18(1):121–129, 2013.

[166] Carl Zimmer. From ants to people. `http://www.nytimes.com/2007/11/13/science/13traff.html?_r=0`, November 2007. [Online; accessed 13-Apr-2016].

APPENDIX

A. ENVIRONMENT SETUP

NOTE: Python3.2 installed by default on Ubuntu 12.10 version 3.3 on 13.04

A.1 MATPLOTLIB

```
sudo apt-get build-dep python-matplotlib
sudo apt-get install python3-dev
wget https://github.com/matplotlib/matplotlib/zipball/master
unzip master
cd matplotlib-matplotlib-???????
sudo python3 setup.py build
sudo python3 setup.py install
sudo apt-get install python3-pyqt4
```

A.2 PYGAME

```
sudo apt-get install python3-dev libjpeg-dev libpng-dev libavformat-dev libsdl-image1.2-d
libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsdl1.2-dev libsmpeg-dev python3-numpy
python-numpy subversion libportmidi-dev libswscale-dev pgf
svn co svn://seul.org/svn/pygame/trunk pygame
cd pygame
python3 setup.py build
sudo python3 setup.py install
```

A.3 PYTHON EDITOR

(Komodo) Download and install from site instructions

```
http://www.activestate.com/komodo-edit/downloads
```

A.4 MySQL

```
sudo apt-get install mysql-server mysql-client
sudo apt-get install apache2 php5 php5-mysql php5-mcrypt
sudo apt-get install phpmyadmin
```

B. PYSWARMWORLD CODE LISTING

B.1 Graphical Environment

```
1  #!/usr/bin/env python3
2  from World import *
3  from Swarm import *
4  from Obstacle import *
5  from Destination import *
6  from BoidSwarm import *
7  from WsnSwarm import *
8  from Globals import *
9  import sys
10
11 status = Globals()
12 myWorld = World(status.X,status.Y,"crash.jpg")
13 myWorld.start()
14
15 destination = Destination(myWorld.screen.get_size()[0]/2,myWorld.screen.get_size()[1]/2, True)
16
17 myWorld.pushDestination(destination)
18
19 status.running = False
20 status.total_iterations = 0
21
22 myWorld.intro()
23
24 while True:
25
26     status.time_passed = myWorld.clock.tick(status.frameRate)
27     status.time_passed_seconds = status.time_passed / status.systemSpeed
28     status.frame = status.getFrame()
29
30     for event in pygame.event.get():
31         if event.type == QUIT:
32             exit()
33
34         if event.type == MOUSEBUTTONDOWN:
35             if event.button == 1:
36                 x, y = event.pos
37                 if status.inputMode == 0:
38                     if myWorld.getSwarmSize() < status.maxParticipants:
39                         if myWorld.collision(x, y):
40                             print("Participant Collision at %s,%s" % (x, y))
41                         else:
42                             participant = Participant(x, y, status.neighbourDistance, status.
    minDistance, status.maxDistance, status.maxSpeed, status.sensorRange)
43                             myWorld.swarms[0].pushBot(participant)
44                             print("ADDED - Participant [%s]-([%s][%s]) Total - [%s]" % (
    participant.id, participant.x, participant.y, myWorld.getSwarmSize()))
45                     else:
```

```
46                          print("MAX Participants for PySwarm Reached (%s) (NOTE: Includes
      Deleted and Killed!)" % (status.maxParticipants))
47                  elif status.inputMode == 1:
48                      destination = Destination(x, y)
49                      myWorld.pushDestination(destination)
50                      print("ADDED - Destination [%s,%s][%s]" % (destination.x, destination.y,
      destination.id))
51                  else:
52                      obstacle = Obstacle(x, y, status.obstacleRepulsion)
53                      myWorld.pushObstacle(obstacle)
54                      print("ADDED - Obstacle [%s,%s][%s]" % (obstacle.x, obstacle.y, obstacle.id
      ))
55
56
57              if event.button == 3:
58                  if status.inputMode == 0:
59                      if myWorld.removeParticipant():
60                          print("Participant Removed!")
61                  elif status.inputMode == 1:
62                      if myWorld.removeDestination():
63                          print("Destination Removed!")
64                  else:
65                      if myWorld.removeObstacle():
66                          print("Obstacle Removed!")
67
68
69
70          if event.type == KEYDOWN:
71
72              shifted = (pygame.key.get_mods() & KMOD_LSHIFT)
73
74  # DYNAMICS AND PHYSICS KEYS
75              if event.key == K_a:
76                  if shifted:
77                      status.sensorRange += 1
78                      myWorld.swarmSensorRange(status.sensorRange)
79                      print("Sensor Range Increased")
80                  else:
81                      if status.sensorRange > 0:
82                          status.sensorRange -= 1
83                          myWorld.swarmSensorRange(status.sensorRange)
84                          print("Sensor Range Decreased")
85                      else:
86                          print("Sensor Range at 0")
87
88              if event.key == K_1:
89                  if shifted:
90                      status.neighbourDistance += 1
91                      myWorld.swarmNeighbourDistance(status.neighbourDistance)
92                      print("Bot Range Increased")
93                  else:
94                      status.neighbourDistance -= 1
95                      myWorld.swarmNeighbourDistance(status.neighbourDistance)
96                      print("Bot Range Decreased")
97
98              if event.key == K_2:
99                  if shifted:
100                     status.minDistance += 1
101                     myWorld.swarmMinDistance(status.minDistance)
102                     print("Bot Closeness Increased")
103                 else:
104                     status.minDistance -= 1
105                     myWorld.swarmMinDistance(status.minDistance)
106                     print("Bot Closeness Decreased")
```

```
107
108             if event.key == K_3:
109                 if shifted:
110                     if ((status.maxSpeed + 1) < 1000.0):
111                         status.maxSpeed += 1
112                     else:
113                         status.maxSpeed = 1000
114                     myWorld.swarmSpeed(status.maxSpeed)
115                     print("Bot Max Speed Increased")
116                 else:
117                     if status.maxSpeed > 1:
118                         status.maxSpeed -= 1
119                     else:
120                         status.maxSpeed = 0
121                     myWorld.swarmSpeed(status.maxSpeed)
122                     print("Bot Max Speed Decreased")
123
124             if event.key == K_4:
125                 if shifted:
126                     if ((status.obstacleRepulsion + 1) < 1000):
127                         status.obstacleRepulsion += 1
128                     else:
129                         status.obstacleRepulsion = 1000
130                     myWorld.swarmObstacleRepel(status.obstacleRepulsion)
131                     print("Obstacle Repulsion Increased")
132                 else:
133                     if status.obstacleRepulsion > 1:
134                         status.obstacleRepulsion -= 1
135                     else:
136                         status.obstacleRepulsion = 0
137                     myWorld.swarmObstacleRepel(status.obstacleRepulsion)
138                     print("Obstacle Repulsion Decreased")
139
140             if event.key == K_5:
141                 if shifted:
142                     if status.processing:
143                         print("Cannot change Sample Rate when Processing")
144                     else:
145                         if (status.sampleRate + 1) > status.frameRate:
146                             status.sampleRate = status.frameRate
147                             print("Sample Rate at Maximum")
148                         else:
149                             status.sampleRate += 1
150                             print("Sample Rate Increased")
151                 else:
152                     if status.processing:
153                         print("Cannot change Sample Rate when Processing")
154                     else:
155                         if status.sampleRate > 1:
156                             status.sampleRate -= 1
157                             print("Sample Rate Decreased")
158                         else:
159                             print("Sample Rate Must be at least 1")
160
161             if event.key == K_6:
162                 if shifted:
163                     if status.processing:
164                         print("Cannot change Frame Rate when Processing")
165                     else:
166                         status.frameRate += 1
167                         status.frame = 0
168                         print("Frame Rate Increased")
169                 else:
170                     if status.processing:
```

```
171                         print("Cannot change Frame Rate when Processing")
172                     else:
173                         if status.frameRate > 1:
174                             status.frameRate -= 1
175                             if status.sampleRate > status.frameRate:
176                                 status.sampleRate = status.frameRate
177                             status.frame = 0
178                             print("Frame Rate Decreased")
179                         else:
180                             print("Frame Rate Must be at least 1")
181 
182             if event.key == K_7:
183                 if shifted:
184                     status.physicsFlyTowardsCentre += 1
185                     print("Physics Fly Way Increased")
186                 else:
187                     status.physicsFlyTowardsCentre -= 1
188                     print("Physics Fly Way Decreased")
189 
190             if event.key == K_8:
191                 if shifted:
192                     status.physicsMoveAway += 1
193                     print("Physics Repulsion Increased")
194                 else:
195                     status.physicsMoveAway -= 1
196                     print("Physics Repulsion Decreased")
197 
198             if event.key == K_9:
199                 if shifted:
200                     status.physicsAvoidObstacle += 1
201                     print("Physics Obstacle Repulsion Increased")
202                 else:
203                     status.physicsAvoidObstacle -= 1
204                     print("Physics Obstacle Repulsion Decreased")
205 
206             if event.key == K_0:
207                 if shifted:
208                     status.physicsMoveTowardsDestination += 1
209                     print("Physics Destination Cohesion Increased")
210                 else:
211                     status.physicsMoveTowardsDestination -= 1
212                     print("Physics Destination Cohesion Decreased")
213 
214             if event.key == K_MINUS:
215                 if shifted:
216                     status.physicsCompressConcave += 1
217                     print("Concave Compress Increased")
218                 else:
219                     status.physicsCompressConcave -= 1
220                     print("Concave Compress Decreased")
221 
222             if event.key == K_r:
223                 if shifted:
224                     status.neighbourRender = not status.neighbourRender
225                     if status.neighbourRender:
226                         print("NEIGHBOUR RENDER ENABLED")
227                     else:
228                         print("NEIGHBOUR RENDER DISABLED")
229                 else:
230                     status.rangeRender = not status.rangeRender
231                     if status.rangeRender:
232                         print("RANGE RENDER ENABLED")
233                     else:
234                         print("RANGE RENDER DISABLED")
```

```
235
236             if event.key == K_v:
237                 status.sensorRender = not status.sensorRender
238                 if status.sensorRender:
239                     print("SENSOR RENDER ENABLED")
240                 else:
241                     print("SENSOR RENDER DISABLED")
242
243             if event.key == K_z:
244                 status.compressConcave = not status.compressConcave
245                 if status.compressConcave:
246                     print("COMPRESS ENABLED")
247                 else:
248                     print("COMPRESS DISABLED")
249
250             if event.key == K_x:
251                 if status.perimeter == status.perimeterMax:
252                     status.perimeter = 0
253                 else:
254                     status.perimeter += 1
255                 if status.perimeter > 0:
256                     print("PERIMETER ENERGY ENABLED")
257                 else:
258                     print("PERIMETER ENERGY DISABLED")
259
260     # OPERATION KEYS
261             if event.key == K_i:
262                 if status.instructions == 2:
263                     status.instructions = 0
264                 else:
265                     status.instructions += 1
266                 if status.instructions > 0:
267                     print("INSTRUCTIONS ENABLED")
268                 else:
269                     print("INSTRUCTIONS DISABLED")
270
271             elif event.key == K_n:
272                 if status.screenMode == 1:
273                     status.screenMode = 0
274                     print("SCREEN NORMAL")
275                 else:
276                     status.screenMode += 1
277                     print("SCREEN NEGATED")
278
279             elif event.key == K_e:
280                 if status.processing:
281                     print("CANNOT CHANGE ENERGY CONSUMPTION LOGGING DURING PROCESSING")
282                 else:
283                     status.energyRecording = not status.energyRecording
284                     if status.energyRecording:
285                         logDate = datetime.datetime.today().strftime("%Y%m%d-%H%M%S")
286                         status.startTime = 0;
287                         status.logCount = 0;
288                         status.log.start(logDate,"Swarm.sql")
289                         print("RECORDING ENERGY CONSUMPTION")
290                     else:
291                         logDate = datetime.datetime.today().strftime("%Y%m%d-%H%M%S")
292                         status.log.finish(logDate)
293                         print("RECORDING ENERGY CONSUMPTION STOPPED")
294
295             elif event.key == K_p:
296                 if divmod(status.frameRate,status.sampleRate)[1] == 0:
297                     status.processing = not status.processing
298                     if status.processing:
```

```
299                         print("PROCESSING ENABLED")
300                     else:
301                         print("PROCESSING DISABLED")
302                 else:
303                     print("ERROR : Frame Rate must be divisable by Sample Rate")
304 
305             elif event.key == K_b:
306                 status.background = not status.background
307                 if status.background:
308                     print("BACKGROUND ENABLED")
309                 else:
310                     print("BACKGROUND DISABLED")
311 
312             elif event.key == K_f:
313                 status.finiteMachine = not status.finiteMachine
314                 if status.finiteMachine:
315                     print("FINITE ENABLED")
316                 else:
317                     print("FINITE DISABLED")
318 
319             elif event.key == K_m:
320                 if status.inputMode == 2:
321                     status.inputMode = 0
322                 else:
323                     status.inputMode += 1
324 
325             elif event.key == K_c:
326                 myWorld.clearDestinations()
327                 myWorld.clearObstacles()
328                 myWorld.clearSwarms()
329                 print("SWARM RESET")
330 
331             elif event.key == K_q:
332                 exit()
333 
334             elif event.key == K_l:
335                 status.processing = False
336                 print("Load World\n")
337                 print("Available Worlds\n")
338                 print("==========\n")
339                 print(myWorld.worldList())
340                 print("==========\n")
341                 worldName = input("World Name:")
342                 if len(worldName) > 1:
343                     if myWorld.loadWorld(worldName):
344                         print("World Loaded!")
345                     else:
346                         print("World Not Found!")
347                 else:
348                     print("Loading World Cancelled!")
349 
350 
351             elif event.key == K_s:
352                 status.processing = False
353                 print("Save World (NOTE: Will overwrite existing world!)\n")
354                 print("Available Worlds\n")
355                 print("==========\n")
356                 print(myWorld.worldList())
357                 print("==========\n")
358                 worldName = input("World Name:")
359                 if len(worldName) > 1:
360                     if myWorld.saveWorld(worldName):
361                         print("World Saved!")
362                     else:
```

```
363                         print("Error Saving World!")
364                 else:
365                     print("Saving World Cancelled!")
366 
367             elif event.key == K_g:
368                 if myWorld.snapshot():
369                     print("Snapshot Taken!")
370                 else:
371                     print("Snapshot Failed!")
372 
373             pygame.event.clear()
374 
375     myWorld.process()
376     if (status.processing == False or (status.processing == True and status.render == True)):
377         myWorld.render()
```

Listing B.1: Graphical environment

B.2 CLI Simulator

```
1  #!/usr/bin/env python3
2  
3  from World import *
4  from Swarm import *
5  from Obstacle import *
6  from Destination import *
7  from BoidSwarm import *
8  from Globals import *
9  import sys, getopt
10 
11 
12 def main():
13     myWorld = World()
14     status = Globals()
15 
16     verbose = False
17     runTime = 0
18     worldName = "X"
19     totalTime = 0.0
20     compressConcave = 0
21     perimeter = 0
22     physicsMoveTowardsDestination = 0
23     physicsFlyTowardsCentre = 0
24     physicsMoveAway = 0
25     physicsAvoidObstacle = 0
26     physicsMoveTowardsDestination = 0
27     physicsCompressConcave = 0
28     frameRate = 0
29     neighbourDistance = 0
30     minDistance = 0
31     maxSpeed = 0
32 
33     status.total_iterations = 0
34     status.compressConcave = False
35     status.startTime = 0;
36     status.energyRecording = True
37     status.processing = True
38     status.finiteMachine = True
39 
```

```
40      try:
41          opts, args = getopt.getopt(sys.argv[1:], "vhw:c:p:t:s:m:n:C:R:o:O:D:V:S:", ["help"])
42      except getopt.GetoptError as err:
43          # print help information and exit:
44          print(err) # will print something like "option -a not recognized"
45          usage()
46          sys.exit(2)
47
48      for o, a in opts:
49          if o == "-v":
50              verbose = True
51          elif o in ("-h","--help"):
52              usage()
53              sys.exit(0)
54          elif o in ("-w"):
55              worldName = a
56
57      print("VERSION: %s" % status.version)
58
59      if not myWorld.loadWorld(worldName):
60          print("+-------------------------+")
61          print("| Error World Not Found!|")
62          print("+-------------------------+\n")
63          usage()
64          exit(0)
65      print("+-------------------------+")
66      print("|      Loaded World       |")
67      print("+-------------------------+")
68      print("%s Loaded!" % (worldName))
69      print("%s Bots" % myWorld.getSwarmSize())
70      print("%s Destinations" % myWorld.getDestinationSize())
71      print("%s Obstacles" % myWorld.getObstacleSize())
72
73      compressConcave = status.compressConcave
74      perimeter = status.perimeter
75      physicsMoveTowardsDestination = status.physicsMoveTowardsDestination
76      physicsFlyTowardsCentre = status.physicsFlyTowardsCentre
77      physicsMoveAway = status.physicsMoveAway
78      physicsAvoidObstacle = status.physicsAvoidObstacle
79      physicsMoveTowardsDestination = status.physicsMoveTowardsDestination
80      physicsCompressConcave = status.physicsCompressConcave
81      frameRate = status.frameRate
82      maxSpeed = status.maxSpeed
83      neighbourDistance = status.neighbourDistance
84      minDistance = status.minDistance
85
86      for o, a in opts:
87          if o in ("-c"):
88              compressConcave = int(a)
89          elif o in ("-p"):
90              perimeter = int(a)
91          elif o in ("-s"):
92              frameRate = int(a) # sample rate
93          elif o in ("-t"):
94              runTime = int(a)
95          elif o in ("-n"):
96              neighbourDistance = int(a)
97          elif o in ("-o"):
98              obstacleRepulsion = int(a)
99          elif o in ("-m"):
100             minDistance = int(a)
101         elif o in ("-D"):
102             physicsMoveTowardsDestination = int(a)
103         elif o in ("-C"):
```

```
104                 physicsFlyTowardsCentre = int(a)
105             elif o in ("-R"):
106                 physicsMoveAway = int(a)
107             elif o in ("-O"):
108                 physicsAvoidObstacle = int(a)
109             elif o in ("-V"):
110                 physicsCompressConcave = int(a)
111             elif o in ("-S"):
112                 maxSpeed = int(a)
113
114     status.compressConcave = compressConcave
115
116     status.perimeter = perimeter
117     status.physicsMoveTowardsDestination = physicsMoveTowardsDestination
118     status.physicsFlyTowardsCentre = physicsFlyTowardsCentre
119     status.physicsMoveAway = physicsMoveAway
120     status.physicsCompressConcave = physicsCompressConcave
121     status.physicsAvoidObstacle = physicsAvoidObstacle
122     status.obstacleRepulsion = obstacleRepulsion
123     status.frameRate = frameRate
124     status.maxSpeed = maxSpeed
125     status.neighbourDistance = neighbourDistance
126     status.minDistance = minDistance
127     status.minDistance = minDistance
128
129     status.time_passed_seconds = status.frameRate / 100
130
131     myWorld.swarmObstacleRepel(status.obstacleRepulsion)
132     myWorld.swarmSpeed(status.maxSpeed)
133     myWorld.swarmNeighbourDistance(status.neighbourDistance)
134     myWorld.swarmMinDistance(status.minDistance)
135
136     print("+------------------------+")
137     print("| Simulation Parameters |")
138     print("+------------------------+")
139     print("Compress - %s" % (status.compressConcave))
140     print("GPS - %s" % (status.perimeterName[status.perimeter]))
141     print("Sample Rate - %ss" % (status.time_passed_seconds))
142     print("Min Distance - %s Units" % (status.minDistance))
143     print("Neighbour Range - %s Units" % (status.neighbourDistance))
144     print("Obstacle Range - %s Units" % (status.obstacleRepulsion))
145     print("==================================")
146     print("Destination Physics - %s" % (status.physicsMoveTowardsDestination))
147     print("Cohesion Physics - %s" % (status.physicsFlyTowardsCentre))
148     print("Repulsion Physics - %s" % (status.physicsMoveAway))
149     print("Obstacle Physics - %s" % (status.physicsAvoidObstacle))
150     print("Compression Physics - %s" % (status.physicsCompressConcave))
151     print("==================================")
152
153     if runTime:
154         print("Duration - %ss" % (runTime))
155         print("==================================")
156
157     logDate = datetime.datetime.today().strftime("%Y%m%d-%H%M%S")
158     status.logCount = 0;
159     status.log.start(logDate,"Swarm.sql")
160
161     while status.processing == True:
162         totalTime += status.time_passed_seconds
163 #         status.frame = status.getFrame()
164         print(".",end="")
165         sys.stdout.flush()
166         myWorld.process()
167         if runTime > 0:
```

```
168                 if (totalTime > runTime):
169                     status.processing = False
170
171     logDate = datetime.datetime.today().strftime("%Y%m%d-%H%M%S")
172     status.log.finish(logDate)
173     print()
174     print("======== COMPLETE ========")
175
176
177 def usage():
178     print("Usage:%s [OPTIONS]" % (os.path.basename(__file__)))
179     print("-c <compress>")
180     print("-p <perimeter>")
181     print("-w <world>")
182     print("-t <time>")
183     print("-d <destination physics>")
184     print("-s <sample rate>")
185     print("-n <neighbour distance>")
186     print("-m <minimum distance>")
187     print("-v Verbose")
188     print("-h Help")
189     print("-D <physicsMoveTowardsDestination>")
190     print("-C <physicsFlyTowardsCentre>")
191     print("-R <physicsMoveAway>")
192     print("-O <physicsAvoidObstacle>")
193     print("-V <physicsCompressConcave>")
194     print("-S <maxSpeed>")
195     print("\ne.g. python3 ./PySwarmWorldCLI.py -w TESTBEDSWARMBIG -c 0 -D 0 -C 5 -R 15 -s 10 -p
          2 -n 60 -m 40 -S 20 -t 65\n")
196
197 if __name__ == "__main__":
198     main()
```

Listing B.2: CLI Environment

C. APPENDIX 3

C.1 Analysis database schema

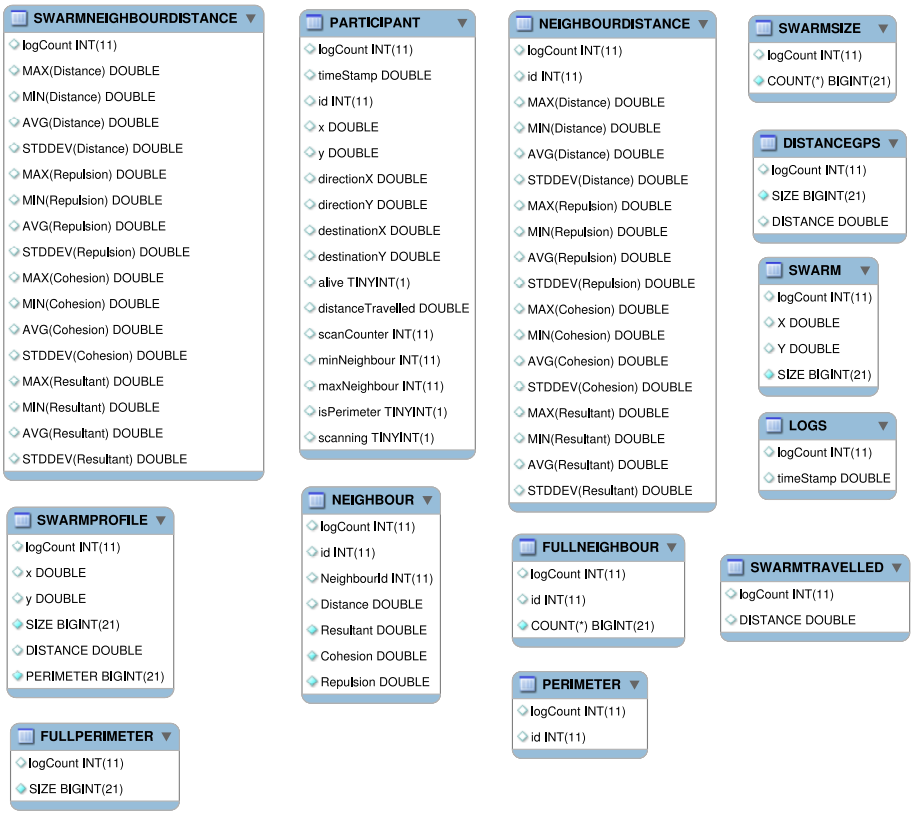


Fig. C.1: Analysis database schema (generated by `PHPMyAdmin 4.5.4.1deb2ubuntu1`)

D. SIMULATOR DATA SETS

```
CREATE TABLE `PARTICIPANT` (
  `logCount` int(11) DEFAULT NULL,
  `timeStamp` double DEFAULT NULL,
  `id` int(11) DEFAULT NULL,
  `x` double DEFAULT NULL,
  `y` double DEFAULT NULL,
  `directionX` double DEFAULT NULL,
  `directionY` double DEFAULT NULL,
  `destinationX` double DEFAULT NULL,
  `destinationY` double DEFAULT NULL,
  `alive` tinyint(1) DEFAULT NULL,
  `distanceTravelled` double DEFAULT NULL,
  `scanCounter` int(11) DEFAULT NULL,
  `minNeighbour` int(11) DEFAULT NULL,
  `maxNeighbour` int(11) DEFAULT NULL,
  `isPerimeter` tinyint(1) DEFAULT NULL,
  `scanning` tinyint(1) DEFAULT NULL
);
```

Listing D.1: `PARTICIPANT` table

```
CREATE TABLE `NEIGHBOUR` (
  `logCount` int(11) DEFAULT NULL,
  `id` int(11) DEFAULT NULL,
  `NeighbourId` int(11) DEFAULT NULL,
  `Distance` double DEFAULT NULL,
  `Resultant` double NOT NULL,
  `Cohesion` double NOT NULL,
  `Repulsion` double NOT NULL
);
```

Listing D.2: `NEIGHBOUR` table

```
CREATE TABLE LOGS AS
        SELECT DISTINCT logCount, timeStamp FROM PARTICIPANT;
```

Listing D.3: `LOGS` view

```
CREATE TABLE SWARM AS
        SELECT logCount, AVG(x) AS 'X', AVG(y) AS 'Y', COUNT(*) AS 'SIZE'
    FROM PARTICIPANT
    WHERE alive = 1 GROUP BY logCount;
```

Listing D.4: `SWARM` view

```
CREATE TABLE PERIMETER AS
        SELECT logCount, id
```

```
3         FROM PARTICIPANT
4         WHERE isPerimeter = 1 AND alive = 1;
```

Listing D.5: `PERIMETER` view

```
1  CREATE TABLE FULLPERIMETER AS
2          SELECT LOGS.logCount, COUNT(PERIMETER.id) AS 'SIZE'
3          FROM LOGS LEFT JOIN PERIMETER ON LOGS.logCount = PERIMETER.logCount
4          GROUP BY LOGS.logCount;
```

Listing D.6: `FULLPERIMETER` view

```
1  CREATE TABLE SWARMSIZE AS
2          SELECT logCount, COUNT(*) FROM PARTICIPANT
3          WHERE alive = 1 GROUP BY logCount;
```

Listing D.7: `SWARMSIZE` view

```
1  CREATE TABLE FULLNEIGHBOUR AS
2          SELECT PARTICIPANT.logCount, PARTICIPANT.id, COUNT(*)
3          FROM PARTICIPANT INNER JOIN NEIGHBOUR ON
4                   (PARTICIPANT.logCount = NEIGHBOUR.logCount
5                           AND
6                    PARTICIPANT.id = NEIGHBOUR.id)
7          WHERE alive = 1 GROUP BY PARTICIPANT.logCount, PARTICIPANT.id;
```

Listing D.8: `FULLNEIGHBOUR` view

```
1  CREATE TABLE NEIGHBOURDISTANCE AS
2          SELECT NEIGHBOUR.logCount, NEIGHBOUR.id, MAX(Distance), MIN(Distance),
3                   AVG(Distance), STDDEV(Distance), MAX(Repulsion), MIN(Repulsion),
4                   AVG(Repulsion), STDDEV(Repulsion), MAX(Cohesion), MIN(Cohesion),
5                   AVG(Cohesion), STDDEV(Cohesion), MAX(Resultant), MIN(Resultant),
6                   AVG(Resultant), STDDEV(Resultant)
7          FROM PARTICIPANT JOIN NEIGHBOUR ON PARTICIPANT.id = NEIGHBOUR.id
8          WHERE alive = 1 GROUP BY NEIGHBOUR.logCount, NEIGHBOUR.id;
```

Listing D.9: `NEIGHBOURDISTANCE` view

```
1  CREATE TABLE SWARMTRAVELLED AS
2          SELECT logCount, SUM(distanceTravelled) AS DISTANCE
3          FROM PARTICIPANT GROUP BY logCount;
```

Listing D.10: `SWARMTRAVELLED` view

```
1  CREATE TABLE DISTANCEGPS AS
2          SELECT FULLPERIMETER.logCount, SIZE, DISTANCE
3          FROM FULLPERIMETER JOIN SWARMTRAVELLED ON
4                   FULLPERIMETER.logCount = SWARMTRAVELLED.logCount;
```

Listing D.11: `DISTANCEGPS` view

```
1  CREATE TABLE SWARMPROFILE AS
2          SELECT SWARM.logCount, x, y, SWARM.SIZE AS SIZE, DISTANCE,
3                   DISTANCEGPS.SIZE AS PERIMETER
```

```
4       FROM SWARM JOIN DISTANCEGPS ON
5           SWARM.logCount = DISTANCEGPS.logCount;
```

Listing D.12: `SWARMPROFILE` view

E. SIMULATOR AND DATA CAPTURE

This appendix presents the modelling techniques applied to the simulator and discusses the processes used to analyse the data sets produced by the simulations it also shows the code required to implement the modelling in thw simulator.

E.1 Python Code

E.1.1 Cohesion

An agent's cohesion is implemented in the simulation software as Listing E.1. Line 4 shows the iteration over all the neighbours. Line 5 calculates the sum of the vectors created by the neighbours. Line 6 divides the summed vectors by the number of neighbours. This process generates the cohesion vector for an agent to its neighbours.

```
1 def flyTowardsCentre(participant):
2   newVector = Vector2()
3   if len(participant.neighbours):
4   for bot in participant.neighbours:
5     newVector += Vector2(bot.x,bot.y)
6     newVector = (Vector2(newVector.x / len(participant.neighbours),newVector.y / len(
      participant.neighbours)) - Vector2.from_floats(participant.x,participant.y))
7   return newVector
```

Listing E.1: Cohesion code

E.1.2 Repulsion

Listing E.2 shows the implementation of inter-agent repulsion in the simulation software. This effect reduces the possibility of agents colliding. Line 4 identifies the distance between a neighbour and an agent. This distance is then used to identify if a repulsive vector needs to be generated (Line 6). Lines 7 and 8 calculate the proportional effect of the repulsion that is then applied to an agent.

```
1 def moveAway(participant):
2   newVector = Vector2()
3   for bot in participant.neighbours:
4     distanceVector = Vector2.from_points((bot.x,bot.y),(participant.x,participant.y))
5     distance = distanceVector.get_length()
6     if participant.minimumClearDistance > distance and distance > 0:
7       percentagePush = ((participant.minimumClearDistance - distance) / distance)
8       newVector += distanceVector.normalise() * (participant.minimumClearDistance *
        percentagePush)
9   return newVector
```

Listing E.2: Repulsion code

E.1.2.1 Destination implementation

Listing E.3 shows the implementation of the destination vector. Line 2 identifies if an agent should apply a destination vector. Line 3 generates a vector based on the **`target`**, The **`target`** is the destination that is closest to the agent. The vector manipulation function (**`normalise()`**) is located in the **`Vector`** class. The result of the process is a vector which influences the agent with a directional bias or a null vector.

```
1 def moveTowardsDestination(participant):
2   if participant.isPerimeter:
3     return Vector2.from_points((participant.x, participant.y),(participant.target.x,participant
      .target.y)).normalise()
4   else:
5     return Vector2()
```

Listing E.3: Repulsion code

E.1.2.2 Obstacle avoidance implementation

Listing E.4 shows how the obstacle avoidance vector is implemented in the simulator. If an agent falls within the range of an obstacle it must be repelled. The method takes as parameters a list of the obstacles and an agent. The method then iterates over each of the obstacles (Line 3) and identifies if the agent is within the obstacle's field effect (Line 6). If the agent is within the field effect the fixed repulsion vector for each obstacle the agent is in range of is added (Line 8). The result is a directional vector that must be applied to the agent.

```
1 def avoidObstacles(participant, obstacles):
2   newVector = Vector2()
3   for obstacle in obstacles:
4     distanceVector = Vector2.from_points((participant.x,participant.y),(obstacle.x,obstacle.y))
5     distance = distanceVector.get_length()
6     if distance < obstacle.repelDistance and distance > 0:
```

```
7         tempVector = Vector2(participant.x,participant.y) - Vector2(obstacle.x,obstacle.y)
8         tempVector = tempVector.normalise() * obstacle.repelDistance
9         newVector += tempVector
10    return newVector
```

Listing E.4: Obstacle avoidance code

E.1.2.3 Agent model implementation

Listing E.5 shows how the implementation of the complete physics model for a boid-based swarm is implemented in the simulator. This code is part of the **`BoidSwarm`** class which inherits the **`Swarm`** class. This method is the specialisation for the boid swarm. Method **`calcTrajectory()`** is the only method in the **`BoidSwarm`** class it collates all the cohesion and repulsion vectors for the inter-agent and inter-object interactions and applies the vectors using the weighted model (line 27). On line 11 the simulator status is checked to identify if the simulation is to implement the convex compression algorithm. If convex compression is to be applied then an alternate physics model is applied to the agents. The repulsion vector is still applied but also a **`moveBetweenTwoNeighbours()`** method generates a vector that is applied. This vector is designed to 'smooth' a swarm's perimeter. This will be covered in more detail in Figure 6.3.

```
1  def calcTrajectory(self, participant, obstacles, destinations):
2    status = Globals()
3    v1 = Vector2()
4    v2 = Vector2()
5    v3 = Vector2()
6    v4 = Vector2()
7    v5 = Vector2()
8
9    nextPosition = Vector2()
10
11   if status.compressConcave:
12     if (participant.isPerimeter and participant.concave):
13       v2 = BoidSwarm.moveAway(participant)
14       v5 = BoidSwarm.moveBetweenTwoNeighbours(participant)
15     else:
16       v1 = BoidSwarm.flyTowardsCentre(participant)
17       v2 = BoidSwarm.moveAway(participant)
18       v4 = BoidSwarm.moveTowardsDestination(participant)
19
20   else:
21     v1 = BoidSwarm.flyTowardsCentre(participant)
22     v2 = BoidSwarm.moveAway(participant)
23     v4 = BoidSwarm.moveTowardsDestination(participant)
24
25   v3 = BoidSwarm.avoidObstacles(participant, obstacles)
26
27   nextPosition += (v1 * status.physicsFlyTowardsCentre) + (v2 * status.physicsMoveAway) + (v3 *
         status.physicsAvoidObstacle) + (v4 * status.physicsMoveTowardsDestination) + (v5 * status
       .physicsCompressConcave)
28   participant.destination = Vector2(nextPosition)
```

```
29  participant.direction = Vector2(nextPosition.normalise())
```

Listing E.5: Agent trajectory code

E.2 Swarm simulator object model

Figure E.1 shows the object model underlying the simulator. The `World` class models the Euclidean plane for the simulations. The `World` class is a container object for other classes including lists of swarms, destinations, and obstacles that make up the environment to be modelled. The `Swarm` class implements a list which contains several `Participants` (agents) that constitute a swarm. The `Participant` class models the swarm agents and inherits the `Bot` class which provides the positioning coordinates for the agents. The `Destination` and `Obstacle` classes inherit their coordinates from the `Bot` class. The `Swarm` class implements a generic physics model for the coordination of the agents. The `BoidSwarm` class inherits the `Swarm` class and implements the algorithm for the boid model which includes the physics modelling. The `Logger` class implements a data logging mechanism and is used to capture the simulation data from within the `World` class via the `Global` class which is an implementation of a singleton containing the environmental constraints.

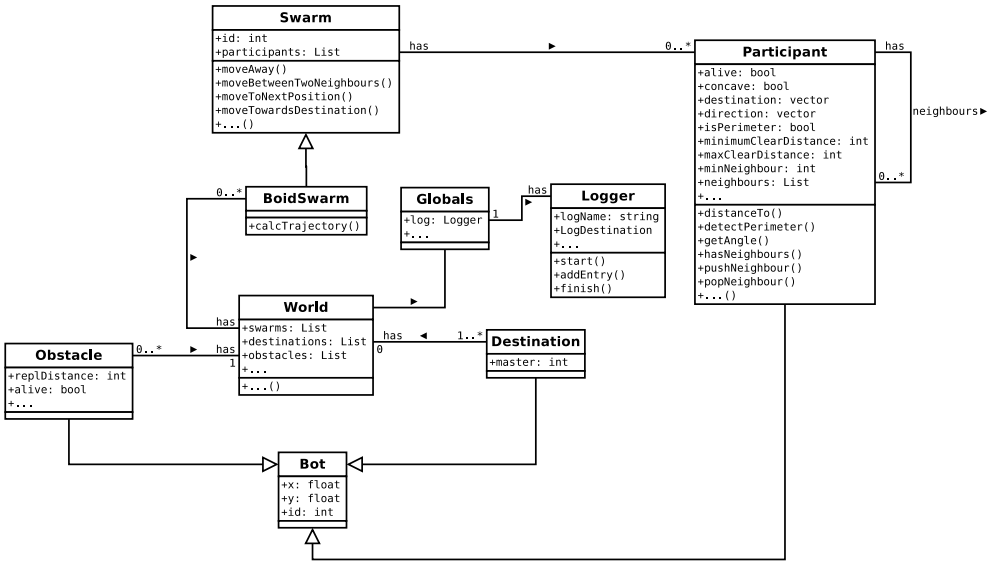


Fig. E.1: Simulator object model

The object modelling shown in Figure E.1 is implemented in Python using packages.

The complete PySwarmWorld application is available at `https://drive.google.com/file/d/0B0DIRBx-V-eYVnNvVEkzUkRuZVk/view?usp=sharing`.

E.3 Simulator data capture

When data logging is activated the simulator generates an SQL log file using the **Logger** Class. As the simulation runs the system, at each time interval (t) calculates all of the agents movements and the magnitudes that affect each agent pair generate. This data is then appended to the SQL log file as a transaction (Listing E.6), shown as (3a) in figure 2.13 on page 28, using the **`addEntry()`** method of the **`Logger`** class. This method generates two SQL insertions that are added for each transaction.

The agent data is added to the log file and it consists of the positional information of the agent along with the destination and directional bias the agent has affecting it. There are also the statistics of the minimum and maximum distances of its neighbours and a boolean flag identifying if the agent is on a perimeter. If an agent is on a perimeter it will be affected by a destination vector. The second set of entries are the neighbour records which include the physics data the neighbour has affecting it through that relationship.

When a simulation is finished the SQL log is closed.

E.4 Data capture implementation

The code in listing E.6 shows how the log entries for a simulation run are captured. Line 1 shows the agent details embedded in an SQL insert statement. This statement is then appended to the log file using the **`addEntry()`** method. The code then iteration over the agents **`bot.neighbours`** list (Line 2). On line 7 the neighbour entries are added to the log file using the **`addEntry()`** method.

```
1 status.log.addEntry("INSERT INTO PARTICIPANT VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s
      ,%s,%s);\n" % (status.logCount,timeStamp,bot.id,bot.x,-bot.y,bot.direction.x,bot.direction
      .y,bot.destination.x,bot.destination.y,bot.alive, bot.distanceTravelled,bot.scanCounter,
      bot.minNeighbour,bot.maxNeighbour,bot.isPerimeter,bot.scanning))
2 for neighbour in bot.neighbours:
3   distanceToNeighbour = Vector2.from_points((bot.x, bot.y),(neighbour.x, neighbour.y))
4   repulsion = (((bot.minimumClearDistance - distanceToNeighbour.get_length()) /
        distanceToNeighbour.get_length()) * bot.minimumClearDistance) * status.physicsMoveAway
5   cohesion = distanceToNeighbour.get_length() * status.physicsFlyTowardsCentre
6   resultant = cohesion - repulsion
```

```
7   status.log.addEntry("INSERT INTO NEIGHBOUR VALUES (%s,%s,%s,%s,%s,%s,%s);\n" % (status.
      logCount, bot.id, neighbour.id, distanceToNeighbour.get_length(), resultant, cohesion, repulsion
      ))
```

Listing E.6: Simulator data capture

E.5 Data capture tables

The tables created from the initial log file are the starting point of the analysis process.

E.5.1 `PARTICIPANT` table (Agents)

The **`PARTICIPANT`** table D.1 is a snapshot of all the agents in the swarm at each time interval. The data that is logged includes the position (**`x`**,**`y`**) of each agent along with the direction of the agent (**`directionX`**, **`directionY`**). The table also includes the field effect values for repulsion (**`minNeighbour`**) and cohesion (**`maxNeighbour`**). The table is used for several aggregations and also to plot the path of individual agents within a simulation.

E.5.2 `NEIGHBOUR` table

The **`NEIGHBOUR`** table D.2 is a snapshot of all the agents that are 'visible' to an agent. This table is directly related to the **`PARTICIPANT`** table via the **`LogCount`** and **`id`** fields as a one to many relationship. This table also contains the calculated magnitudes for the repulsion and cohesion of each neighbour to its parent. The table also includes the calculated distance the neighbour is away from the parent agent.

E.6 Simulator data aggregation

Following the simulation the SQL log file is imported into an 'analysis' database. This is shown as (3b) in figure 2.13 on page 28. Once the base data is loaded into the database the aggregation of the data can be executed.

E.6.1 Data aggregation views

The data aggregation is shown as stage (4) in figure 2.13 on page 28. The aggregated views of the data provide a deeper level of information that is required for the analyse of the internal activity of the swarm.

There are several steps in the aggregation of the base data. Each step builds up a set of persistent views. The views expose the 'dynamics' of the swarm at each time interval. The views are persistent to improve the query time of the simulation analysis. This technique is used as the data, once captured, is static. The data model for the swarm dynamics database is shown in figure C.1 on page 219. The following sections give a breakdown of the aggregation views and their purpose.

E.6.1.1 `LOGS` view

The **`LOG`** view (Listing D.3) is an aggregation of the time intervals in the simulation. It includes a counter which is a sequence number for each time increment in the system and a timestamp in seconds of the current simulation runtime.

E.6.1.2 `SWARM` view

The **`SWARM`** view (Listing D.4) allows the centroid of the swarm to be tracked. This allows the position of a swarm to be monitored based on its centre of mass. The view takes into account agents that have been removed from the swarm by only selecting 'live' agents from the base data. The view also provides a count of the number of live agents that are currently in the swarm.

E.6.1.3 `PERIMETER` view

The **`PERIMETER`** view (Listing D.5) provides a list of the agent identifiers (**`id`**) that have their perimeter status flag (**`isPerimeter`**) set to **`TRUE`** in a specific time slice (**`logCount`**).

E.6.1.4 `FULLPERIMETER` view

The **`FULLPERIMETER`** view (Listing D.6) aggregates the **`PERIMETER`** view to provide a total (**`SIZE`**) of how many agents have their perimeter flag (**`isPerimeter`**) set. These

agents may be influencing a swarms direction. This allows changes in the perimeter size to me monitored as a simulation progresses.

E.6.1.5 SWARMSIZE view

The **SWARMSIZE** view (Listing D.7) provides a running total of the total number of agents that are active in a swarm. This allows the detection of agent loss in a simulation.

E.6.1.6 FULLNEIGHBOUR view

The **FULLNEIGHBOUR** view (Listing D.8) provides a running total, for each time slice, of the number of neighbours each agent has. This allows the expansion phase of a swarm to be monitored. This view can also be used to identify if the cohesion and repulsion weightings and distances allow the swarm to create hexagonal lattices.

E.6.1.7 NEIGHBOURDISTANCE view

The **NEIGHBOURDISTANCE** view (Listing D.9) provides access to the distances between each agent in the swarm and its neighbours. This allows the analysis of the swarms structure based on inter-agent distances. The inter-agent distances are the swarm attribute identified by Navarro and Matía in their swarm metrics paper [104]. The view also provides access to the magnitudes of the cohesion, repulsion and resultant magnitude for each agent-neighbour relationship. These three attributes allow a more detailed view of the state of the swarm to be identified. These attributes highlight a swarms structural 'performance'. This view also includes the maximum and minimum magnitudes and provides a 'picture' of the swarm's distribution which can be realised graphically.

E.6.1.8 SWARMTRAVELLED view

The **SWARMTRAVELLED** view (Listing D.10) provides the total distance travelled by all the agents in the swarm. This information can be used to compare the distance each agent has travelled to the distance the centroid of the swarm has travelled. This creates a measure to identify the efficiency of the perimeter influence on a swarm's directional movement.

E.6.1.9 `DISTANCEGPS` view

The **`DISTANCEGPS`** view (Listing D.11) provides the total distance travelled by agents that have GPS sensors enabled and are coordinating the swarm. This information can be used to identify the degree of GPS influence on the movement of a swarm.

E.6.1.10 `SWARMPROFILE` view

The **`SWARMPROFILE`** view (Listing D.12) identifies the centroid of the swarm at each time interval. It also provides general information about how large the swarm is and the distance travelled by the agents.

A copy of all of the experimental data extracts is available at: `https://drive.google.com/file/d/0B0DIRBx-V-eYTVNQeFVEUFlOMEk/view?usp=sharing`.

E.7 Data graphing tools

The resultant views allow the visualisation of the swarm activity. The data realisation tool used in this thesis is **`matplotlib`** [63]. This is a python-based graphing environment which provides facilities similar to **`MatLab`** [91].

The aggregated data is extracted from the MySQL [23] database using the **`cymsql`** MySQL connector [36] shown as (5) in figure 2.13 on page 28. The data when extracted is applied to a specific plot type. These plots are then rendered to visualise the characteristics of the swarm, this is shown as (6) in (figure 2.13).

The sample code (Listing E.7) is for a simple path plot for a swarm. Line 11 shows the iteration over the records extracted from the database using the **`cymysql`** connector (Line 8). The data is then pushed into two lists (x[ ],y[ ]), shown in (Lines 12 and 13). Line 15 shows the selection of the plot type (**`plot`**) and line 21 generates the graph (**`show`**). An example of the graph this generates, a swarm's centroid path, is shown in figure E.2 on page 232.

```
1 #!/usr/bin/python3
2 import cymysql
3 import matplotlib.pyplot as plt
4
5 x = []
6 y = []
7
```

```
8  conn = cymysql.connect(host='127.0.0.1', user='phd', passwd='*********', db='ST-TEST3-20S',
       charset='utf8')
9  cur = conn.cursor()
10 cur.execute('SELECT * FROM SWARM')
11 for r in cur.fetchall():
12     x.append(r[1])
13     y.append(-r[2])
14
15 p = plt.plot(x,y, label="Baseline",color="k", linewidth=1)
16 plt.title('Swarm Path Propagation (200)',fontsize=30)
17 plt.ylabel('y',fontsize=30)
18 plt.xlabel('x',fontsize=30)
19 plt.legend(fontsize=30)
20
21 plt.show()
```

Listing E.7: Sample **matplotlib** script

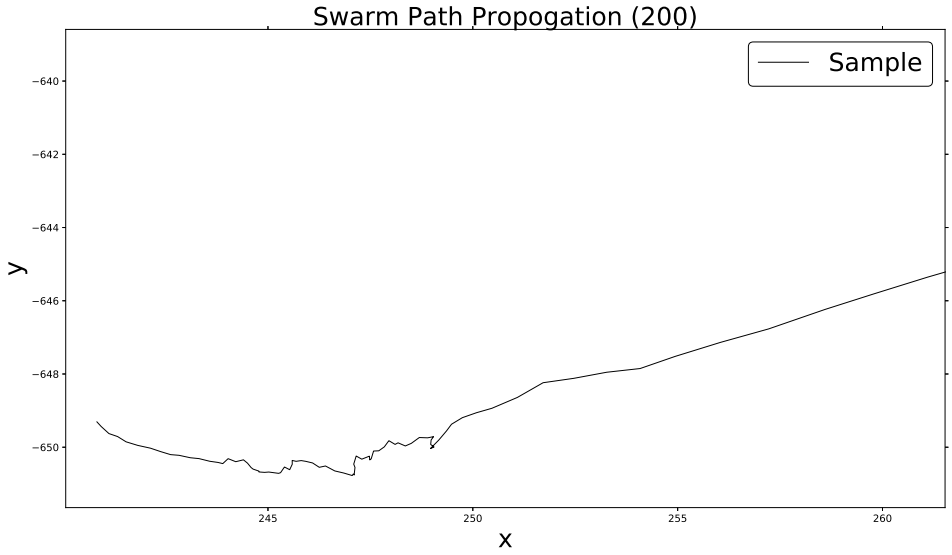


Fig. E.2: Sample **matplotlib** graph

A copy of all of the graphing scripts is available at: `https://drive.google.com/file/d/0B0DIRBx-V-eYVVg1T1JXREx2eGM/view?usp=sharing`.